Data Preparation & Visualisation

Question 1: You must perform appropriate EDA on your dataset, rationalizing and detailing why you chose the specific methods and what insight you gained

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sbn
from IPython.display import display
from scipy import stats
from scipy.stats import binom, poisson
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
import re
import warnings
warnings.filterwarnings('ignore')
```

After importing all the required liabraries I have download the dataset from CSO official website. which is unemployment datase. Lets load it into here so we can perform operations on it. To load our dataset I am using pandas built in method named read_csv, which will load the datset to work it on.

```
monthlyUnemploymentDF=pd.read_csv('MUM01.20231006T231032.csv')
```

Before visualize our data. We need to understrannd it first. for that let's perform some EDA on it. First of all let's look into out data. we can use head method to to look into the data that how it looks.

```
monthlyUnemploymentDF.head()
```

```
                               Statistic Label        Month
Age Group  \
0  Seasonally Adjusted Monthly Unemployment Rate  1998 January  15 -
24 years
1  Seasonally Adjusted Monthly Unemployment Rate  1998 January  15 -
24 years
2  Seasonally Adjusted Monthly Unemployment Rate  1998 January  15 -
24 years
3  Seasonally Adjusted Monthly Unemployment Rate  1998 January  25 -
74 years
4  Seasonally Adjusted Monthly Unemployment Rate  1998 January  25 -
74 years

          Sex UNIT  VALUE In Percentage    UNIT.1  VALUE In Thousands

0  Both sexes    %                 13.3  Thousand                55.4

1        Male    %                 13.7  Thousand                31.5

2      Female    %                 12.7  Thousand                23.9

3  Both sexes    %                  7.4  Thousand                96.2

4        Male    %                  6.9  Thousand                55.4
```

With the help of head method we got to know that our dataset contains these columns:

1)The first column is statistic lable which tellls us the type of statistics. And in this case it is seasonally Adjusted Monthly Unemployment Rate.

2)Second is Month column which contains Month and year of data.

3)Third is Age group, which contains all the age groups

4)The fourth is Ses which contains genders

5)Fifth is unit and it is unit of the 'Value In Percentage Column' and it is %

6)Next column is Value In Percentage which contains unemployment rate in percentage.

7)seventh column is unit of 'Value In Thousands' column and it is in thousand

8)the last column in our dataset is Value In Thousands which contains the unemployment rate in thousands.

We can also use tail method to check last few rows of our data.

```
monthlyUnemploymentDF.tail()
```

```
                                    Statistic Label           Month  \
1849  Seasonally Adjusted Monthly Unemployment Rate  2023 September
1850  Seasonally Adjusted Monthly Unemployment Rate  2023 September
1851  Seasonally Adjusted Monthly Unemployment Rate  2023 September
1852  Seasonally Adjusted Monthly Unemployment Rate  2023 September
1853  Seasonally Adjusted Monthly Unemployment Rate  2023 September

          Age Group         Sex UNIT  VALUE In Percentage    UNIT.1  \
1849  15 - 24 years        Male    %                 12.5  Thousand
1850  15 - 24 years      Female    %                 11.3  Thousand
1851  25 - 74 years  Both sexes    %                  3.1  Thousand
1852  25 - 74 years        Male    %                  3.2  Thousand
1853  25 - 74 years      Female    %                  2.9  Thousand

      VALUE In Thousands
1849                22.5
1850                19.0
1851                74.2
1852                40.7
1853                33.5
```

To check how many roas and column do we have in our dataset, we can use shape mehtod. this way we will get the idea that how large our dataset is. we will get total nunber of rows and columns in return.

```
monthlyUnemploymentDF.shape

(1854, 8)
```

Now lets check for the colunm names of my dataset. We can use columns keyword for that. it will retuen all the colunm names to us.

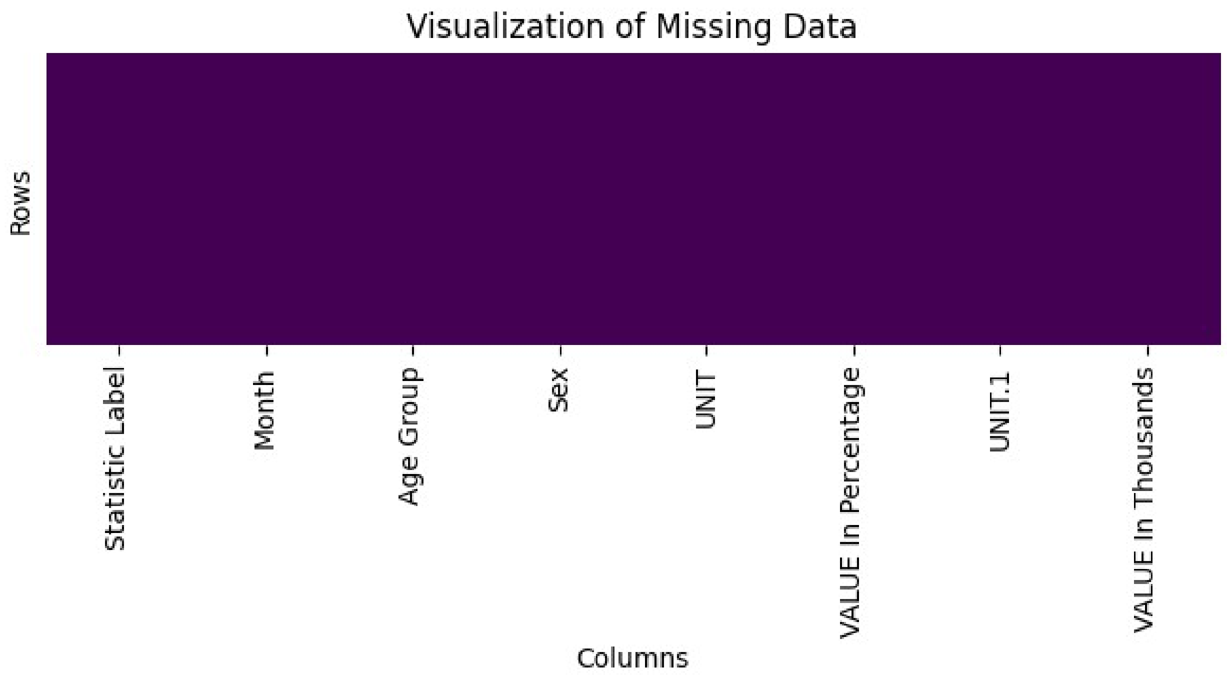
```
monthlyUnemploymentDF.columns

Index(['Statistic Label', 'Month', 'Age Group', 'Sex', 'UNIT',
       'VALUE In Percentage', 'UNIT.1', 'VALUE In Thousands'],
      dtype='object')
```

Now Let's look for any Null values in our dataset. we use isNull method to look for null values in our datase. We will use sum method with it. Sum will count all the missing values and show it to us.This is important for the analysis beause if we have null values so we need to fix it using the mean or median.

```
monthlyUnemploymentDF.isnull().sum()

Statistic Label        0
Month                  0
Age Group              0
Sex                    0
UNIT                   0
VALUE In Percentage    0
UNIT.1                 0
VALUE In Thousands     0
dtype: int64
```

Now let's graphically display the data as well to see if we have any missing data.

```
plt.figure(figsize=(8,2))
sbn.heatmap(monthlyUnemploymentDF.isnull(),cbar=False,cmap='viridis',
yticklabels=False)
plt.title("Visualization of Missing Data")
plt.xlabel("Columns")
plt.ylabel("Rows")
plt.show()
```



As we can see above that we don't have any missing data in our dataset

Now lets look for any duplicate values in our datase. It is very common to get duplicate data in our dataset.It is important step to look for any duplicate rows in dataset and remove them if we find any.

```
dupliatevaluesForDA = monthlyUnemploymentDF.duplicated().sum()
print("Duplicate rows in our Dataset are: ", dupliatevaluesForDA)

Duplicate rows in our Dataset are:  0
```

Anoother import step is to check the data type of each column. and if there is a mismath in the data aand it's type. we can fix it. Below I used the info method to test the data types of each column.

```
monthlyUnemploymentDF.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1854 entries, 0 to 1853
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Statistic Label      1854 non-null   object 
 1   Month                1854 non-null   object 
 2   Age Group            1854 non-null   object 
 3   Sex                  1854 non-null   object 
 4   UNIT                 1854 non-null   object 
 5   VALUE In Percentage  1854 non-null   float64
 6   UNIT.1               1854 non-null   object 
 7   VALUE In Thousands   1854 non-null   float64
dtypes: float64(2), object(6)
memory usage: 116.0+ KB
```

As we noticed above that our Month column is of object type. We will convert it later while preparing our data for ML.

```
monthlyUnemploymentDF['Month'] =
pd.to_datetime(monthlyUnemploymentDF['Month'])
print("New Date: ", monthlyUnemploymentDF['Month'])

New Date:  0      1998-01-01
1      1998-01-01
2      1998-01-01
3      1998-01-01
4      1998-01-01
          ...
1849   2023-09-01
1850   2023-09-01
1851   2023-09-01
1852   2023-09-01
1853   2023-09-01
Name: Month, Length: 1854, dtype: datetime64[ns]
```

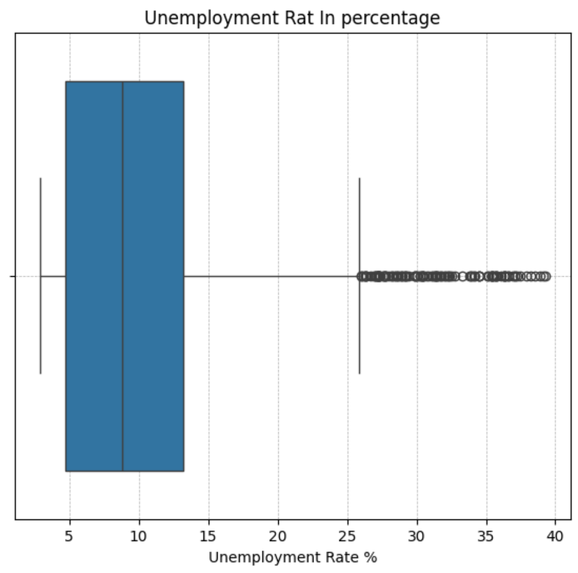
Now lets get some statistics about our data using describe method. It will give us standard deviation,count, mean etc

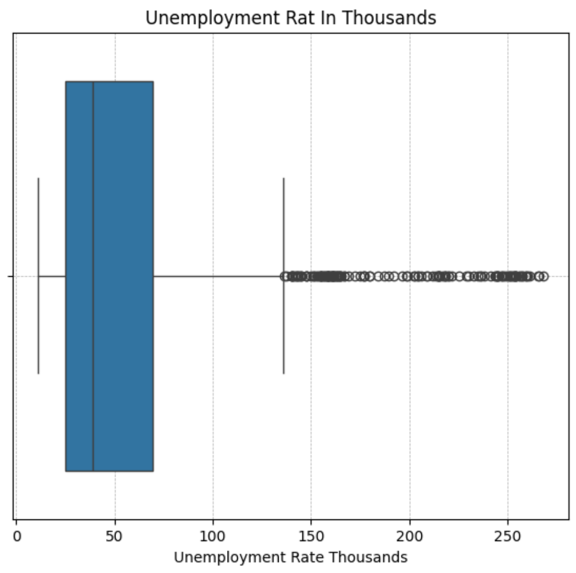
```
monthlyUnemploymentDF.describe()

       VALUE In Percentage  VALUE In Thousands
count          1854.000000         1854.000000
mean             10.680529           56.665696
std               7.539243           48.541094
min               2.900000           11.000000
25%               4.700000           25.200000
50%               8.800000           39.000000
75%              13.200000           69.675000
max              39.300000          268.400000
```

With the help of describe method we came to the conclusion that in Value In Percentage the average unemployment rate is around 10.68% and the minimum unemployment rate is 2.9%. We got discovered that the median is 8.8 percent. and we noticied that half of the observasions are below 8.8% and half are above 8.8% in uneployment rates.The aevrage of unemployment in thousands is 56.67k. The minimum count is 11k and maximum is 268.4k and median is 39k.

Now let's visualize out of range data. we can use numerical columns for that. and we weill be using boxplot to look for outliers. Outliers are data points that are different from other data points. for example we have a data of a store and we are checking how many oranges are sold per day and one datapoint shows that 200,000 sold per day so that will be outlier.

```
# visualizing out of range data data using boxplot
fig,ax=plt.subplots(nrows=1, ncols=2, figsize=(15,6))

# BoxPlot for value In Percentage
sbn.boxplot(x=monthlyUnemploymentDF['VALUE In Percentage'],ax=ax[0])
ax[0].set_title("Unemployment Rat In percentage")
ax[0].set_xlabel("Unemployment Rate %")
ax[0].grid(True, which="both",linestyle='--',linewidth=0.5)

# BoxPlot for value In Thousands
sbn.boxplot(x=monthlyUnemploymentDF['VALUE In Thousands'],ax=ax[1])
ax[1].set_title("Unemployment Rat In Thousands")
ax[1].set_xlabel("Unemployment Rate Thousands")
ax[1].grid(True, which="both",linestyle='--',linewidth=0.5)
```

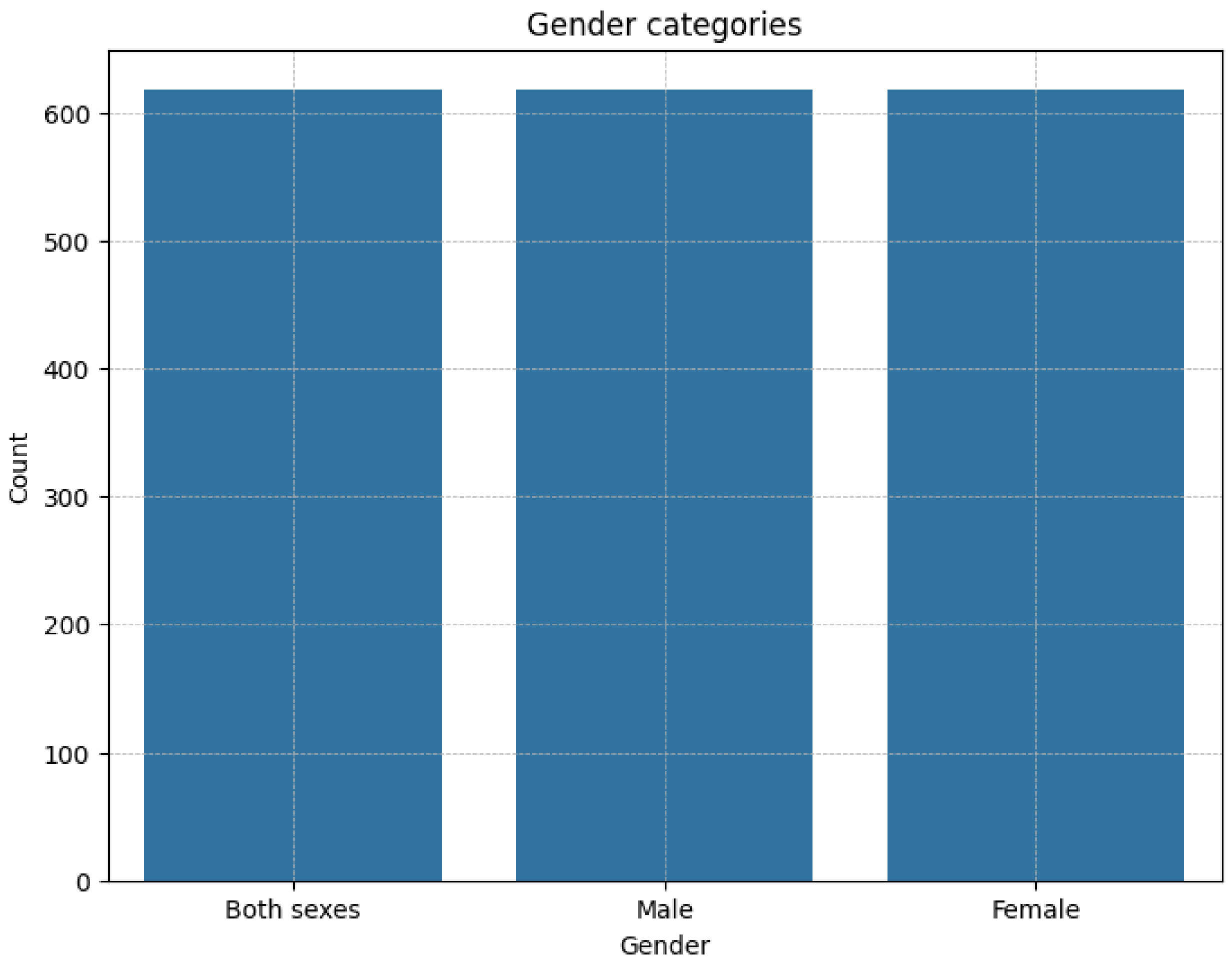




We will not be fixing any outliers for now. Since wqe are dealing with unemployment rate so they could have valuable information.

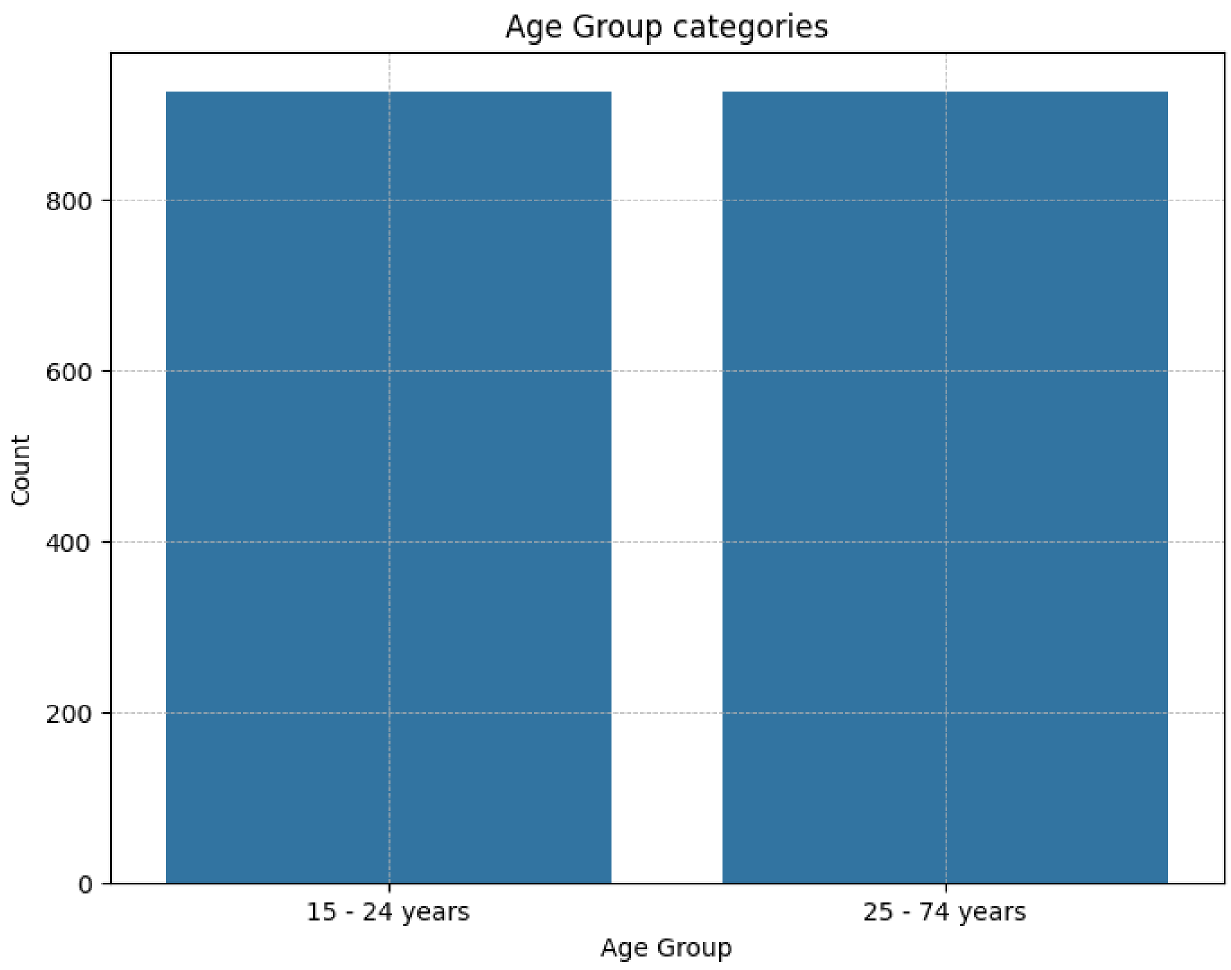
Now let's look for dirty data. let's use some categorical column (Sex and Age Group) and display their distribution to find inconsistencies. Let's test Sex column and look for inconsistant labels or uneexpected categories.

```python
#visualize the distribution of categories in the sex column
plt.figure(figsize=(8,6))
sbn.countplot(data=monthlyUnemploymentDF,x='Sex')
plt.title("Gender categories")
plt.xlabel("Gender")
plt.ylabel("Count")
plt.grid(True,which="both",linestyle="--",linewidth=0.5)
plt.show()
```



The above plot shows that the data is consistant. We got distinct categories of male, female and both sexes. Now to futher check for dirty data we should also check the other categorical column 'Age Group' to make sure that our data isn't dirty. So let's check the other column now.

```
#visualize the distribution of categories in the sex column
plt.figure(figsize=(8,6))
sbn.countplot(data=monthlyUnemploymentDF,x='Age Group')
plt.title("Age Group categories")
plt.xlabel("Age Group")
plt.ylabel("Count")
plt.grid(True,which="both",linestyle="--",linewidth=0.5)
plt.show()
```



We got the distinct categories(15 - 24 years and 25 to 75 years). so the data seems consistant.

Now lets do one more step to check the data consistency . As we already checked for any missing values in our dataset so let's check if the month column is in good format.

```
# Extracting unique month year combinations
monthUniqueValues=monthlyUnemploymentDF['Month'].unique()

# let's check for first few and last few rows for consistency
firstAndLastRows=np.concatenate([monthUniqueValues[:5],monthUniqueValu
es[-5:]])
firstAndLastRows
```

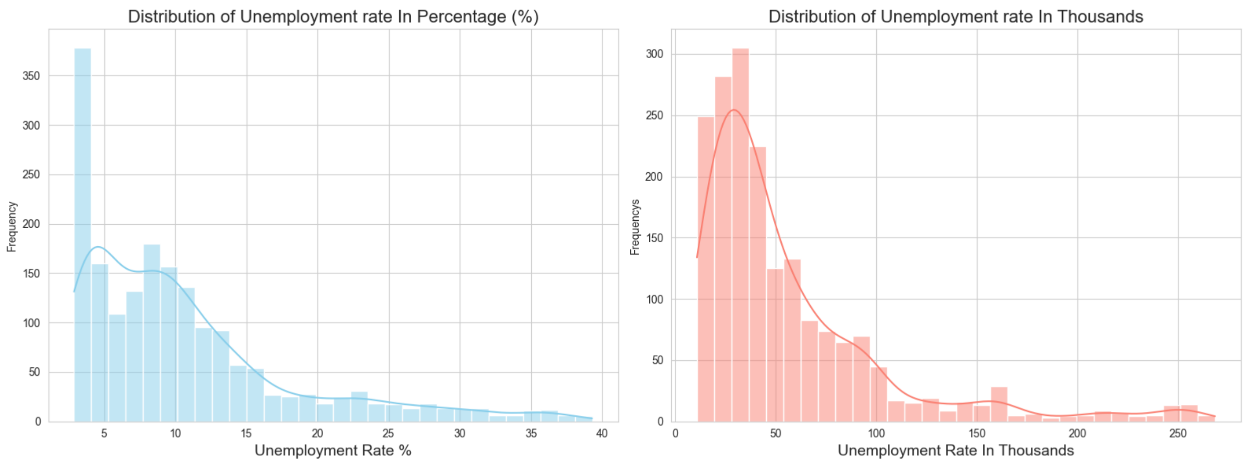
```
array(['1998-01-01T00:00:00.000000000', '1998-02-
01T00:00:00.000000000',
       '1998-03-01T00:00:00.000000000', '1998-04-
01T00:00:00.000000000',
       '1998-05-01T00:00:00.000000000', '2023-05-
01T00:00:00.000000000',
       '2023-06-01T00:00:00.000000000', '2023-07-
01T00:00:00.000000000',
       '2023-08-01T00:00:00.000000000', '2023-09-
01T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

This data looks consistant. the first few rows are of from 1998 January To May. and last few are from 2023 june to september

Lets Visualize our dataset to understand it better. We can use different plots to know more about our data visually

Let's plot the distribution of 'Vvalue in thousands' and 'Value In pErentage'. I am using histogram to plot it because it is primary tool to understand the numerical data. The data get partitioned into bins and then it counts the number of rows for every bin and this way we can see if data is clustered and if there are any outliers. With histograms we an have quick view of distributions

```python
monthlyUnemploymentDF['Month'] =
pd.to_datetime(monthlyUnemploymentDF['Month'])
# first of all i will set the style for seaborn plot to get better
aesthetics
sbn.set_style("whitegrid")

# ccreate a figure for 2 subplots. one for each column
fig,ax = plt.subplots(1,2,figsize=(16,6))

# lets plot distribution for 'Value In Percentage'
# i will be using kernel density estimation to get a smooth curve
which estimates the probability density function of the variable
sbn.histplot(monthlyUnemploymentDF['VALUE In Percentage'], bins=30,
ax=ax[0],kde=True,color='skyblue')
ax[0].set_title("Distribution of Unemployment rate In Percentage
(%)",fontsize=16)
ax[0].set_xlabel("Unemployment Rate %",fontsize=14)
ax[0].set_ylabel("Frequency")

# Lets plot for Value In Thousands
sbn.histplot(monthlyUnemploymentDF['VALUE In Thousands'], bins=30,
ax=ax[1],kde=True,color='salmon')
ax[1].set_title("Distribution of Unemployment rate In
Thousands",fontsize=16)
ax[1].set_xlabel("Unemployment Rate In Thousands",fontsize=14)
ax[1].set_ylabel("Frequencys")

plt.tight_layout()
plt.show()
```



well this graph explains taht the most of the data points indicate the unemployment rate between 3 to 15 percent. In few ases the unemployment rate is close to 40 percent. This could be becaus eof specific genders,age groups or year and month. And in unemployment rate in thousands most of the data points are between 10 to 80k. few people crossing 200k and this could be because of different age group or gender or time.

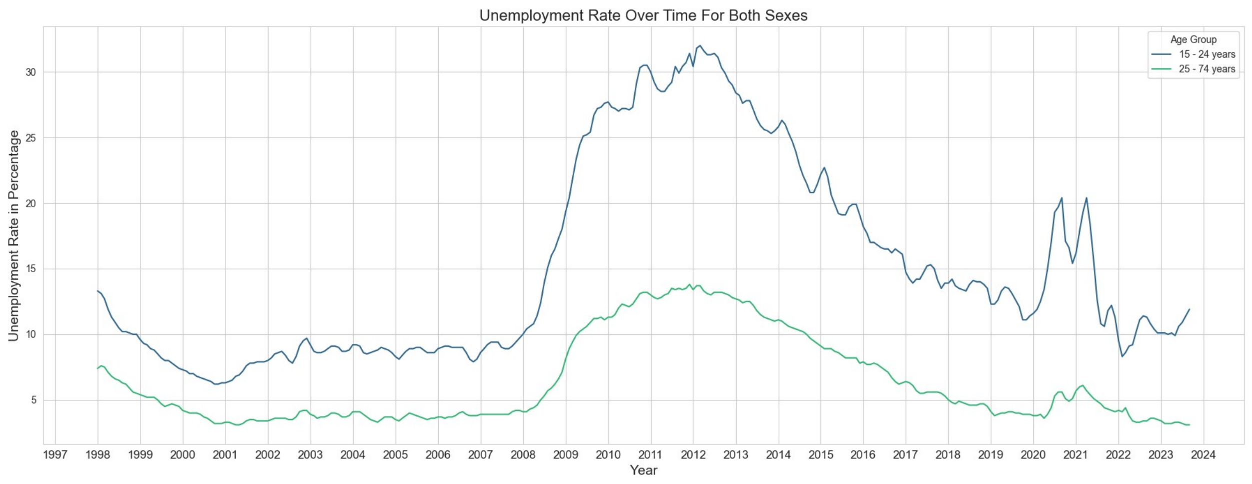
I used the different colors for two graphs because this way it will be differentisted. and viewr an focus on them. skyblue and salmon color really contras well.

Let's plot another graph for Unemployment Rate Over Time and understand our data better

Let's create line graph to show the unemployment rate over time. As we know thes graphs are goood to show change over time. I am going to plot unemployment rate by month so this is best graoh to work with.

```
import matplotlib.dates as mdates
# First of all let's select both sexes from our data so we can see the
graph for ale and female
monthlyUnemploymentDF['Month'] =
pd.to_datetime(monthlyUnemploymentDF['Month'])
filetredOfBothSexForViz =
monthlyUnemploymentDF[monthlyUnemploymentDF['Sex']=='Both sexes']

# let's set teh size of our graph
plt.figure(figsize=(18,7))

# we will draw line to show the unemployment rate in percentage and
diffrent shades for different age groups and with help of viridis we
an easily differentiate colors because it gives the good progression
```

```
of colors
sbn.lineplot(data=filetredOfBothSexForViz, x= 'Month',y='VALUE In
Percentage',hue="Age Group",palette="viridis")

# Let's set the title and labels and also the size
plt.title("Unemployment Rate Over Time For Both Sexes", fontsize=16)
plt.xlabel('Year', fontsize = 14)
plt.ylabel("Unemployment Rate in Percentage", fontsize = 14)
plt.legend(title="Age Group")

# Since we have alot of data and we can't show motnh of eah year
beause it won't be readable so let's adjust our x-axix to show only
start of eery year
plt.xticks(fontsize=12)
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

# we don't want our lables and titles to overlap so lets use
tight_layout function and display the graph
plt.tight_layout()
plt.show()
```

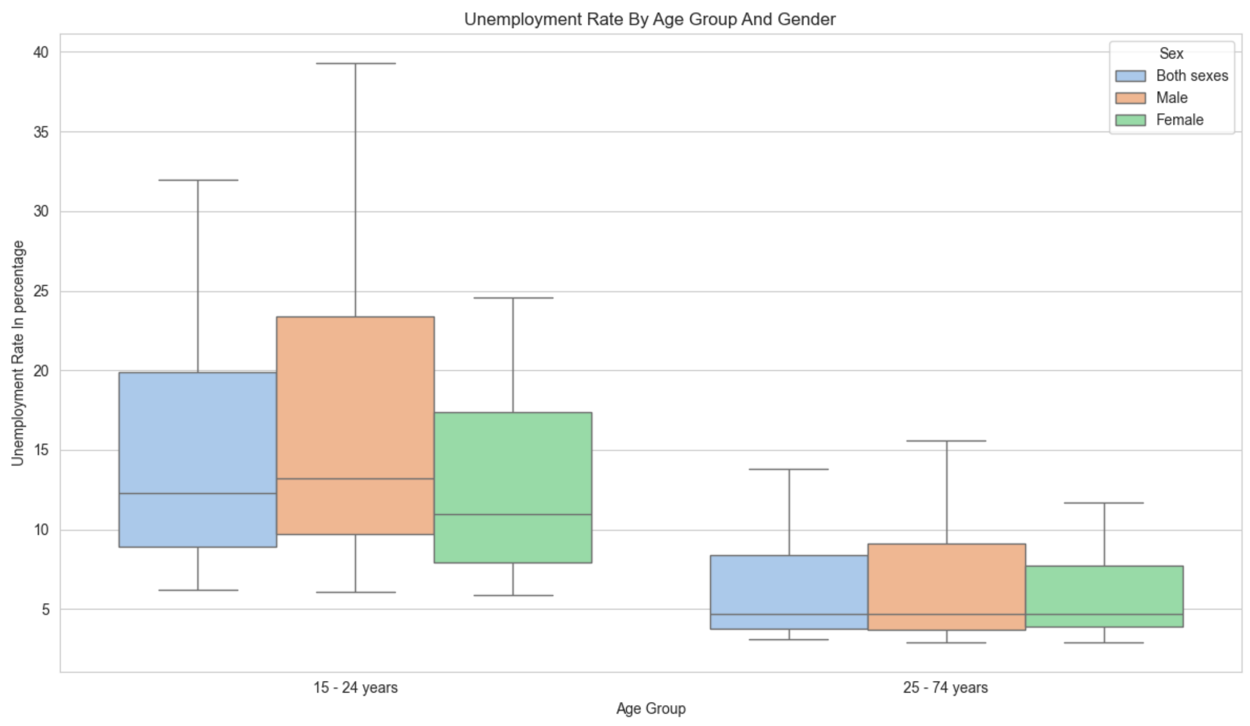


With the help of above graph we can see how the unemployment rate has changed over time for diffrent age groups. With lines in our graph we an see good and bad times in unemployment rate. This graph is for different age groups so we can see if one age group is affected in different time. Also I have increasef the size of labels and titiles so they area easily readable now.

Now let's create one more graph and display unemployment rate by Age groups and Genders.

With teh help of this graph we an get to knwo how unemployment rate changes across different age groups and genders. So we will know which age groups are ore affected by unemployment rate and if tere are inconsistancy between different genders in these age groups.

```
# Let's create size of plot
monthlyUnemploymentDF['Month'] =
pd.to_datetime(monthlyUnemploymentDF['Month'])
plt.figure(figsize=(12,7))
sbn.boxplot(data=monthlyUnemploymentDF,x='Age Group', y='VALUE In
Percentage',hue='Sex',palette='pastel')
plt.title("Unemployment Rate By Age Group And Gender")
plt.xlabel("Age Group")
plt.ylabel("Unemployment Rate In percentage")
plt.legend(title="Sex")
plt.tight_layout()
plt.show()
```



By using Boxplot we can compare the distribution of unemployment rates by different age groups and genders side by side. we can use other plots like bar plots for this situation as well but the boxplot is useful when we want to study the distribution and spread of data.

# Question: You must also rationalise justify and detail all the methods used to prepare the data for ML. [0-30]

As we already checked for any null or duplicate values in our EDA and fortunately we have no null or duplicate alues in our dataset. This is important step because if we have any gaps in our data so we need to fill them or handle duplicate values and we don't do that so we will get incorrect results while using the data for predictions.

Now lets move to the next step and prepare our data for ML

Let's do some feature engineering. As we know that features are used to make predictions. So feature engineering is like making those variables for meaningful analysis. For example in our dataset the Month column has both year and Month in it (like this: 1998 July).Let's get year and month into 2 seprate columns. By doing this it will be easy for Machine Learning models to spot trends over time.

```
# Let's first convert the month into datetime
monthlyUnemploymentDF['Year']=pd.to_datetime(monthlyUnemploymentDF['Mo
nth'])

# now we can extract the month and year using from datetime format
monthlyUnemploymentDF['Year']=monthlyUnemploymentDF['Month'].dt.year
monthlyUnemploymentDF['MonthNumber'] =
monthlyUnemploymentDF['Month'].dt.month

# Now we have month and year seprately. so lets drop month column
monthlyUnemploymentDF.drop('Month',axis=1,inplace=True)
monthlyUnemploymentDF.head()
```

```
                                 Statistic Label      Age Group
Sex  \
0  Seasonally Adjusted Monthly Unemployment Rate  15 - 24 years  Both
sexes
1  Seasonally Adjusted Monthly Unemployment Rate  15 - 24 years
Male
2  Seasonally Adjusted Monthly Unemployment Rate  15 - 24 years
Female
3  Seasonally Adjusted Monthly Unemployment Rate  25 - 74 years  Both
sexes
4  Seasonally Adjusted Monthly Unemployment Rate  25 - 74 years
Male

  UNIT  VALUE In Percentage    UNIT.1  VALUE In Thousands  Year
MonthNumber
0    %                 13.3  Thousand                55.4  1998
1
1    %                 13.7  Thousand                31.5  1998
```

```
1
2    %                 12.7  Thousand                23.9  1998
1
3    %                  7.4  Thousand                96.2  1998
1
4    %                  6.9  Thousand                55.4  1998
1
```

Now let's move further and convert our categorical features into numerial columns becasue we know that machine learning models prefer numbers. We have 2 categorical columns (Age Group and Sex) and limited numbers of unique values so we will use one-hot encoding which is good for such situations. with help of this we can create a column for each category and use 0 and 1 to show the presence of value. This process is called encoding. Machine learning models use such data effetively.

```
encodedDataForAgeAndSex=pd.get_dummies(monthlyUnemploymentDF,
columns=['Age Group', 'Sex'])
#Lets's display to see the output
encodedDataForAgeAndSex

                                    Statistic Label UNIT  VALUE In
Percentage  \
0     Seasonally Adjusted Monthly Unemployment Rate    %
13.3
1     Seasonally Adjusted Monthly Unemployment Rate    %
13.7
2     Seasonally Adjusted Monthly Unemployment Rate    %
12.7
3     Seasonally Adjusted Monthly Unemployment Rate    %
7.4
4     Seasonally Adjusted Monthly Unemployment Rate    %
6.9
...                                             ...  ...
...
1849  Seasonally Adjusted Monthly Unemployment Rate    %
12.5
1850  Seasonally Adjusted Monthly Unemployment Rate    %
11.3
1851  Seasonally Adjusted Monthly Unemployment Rate    %
3.1
1852  Seasonally Adjusted Monthly Unemployment Rate    %
3.2
1853  Seasonally Adjusted Monthly Unemployment Rate    %
2.9
```

```
        UNIT.1  VALUE In Thousands  Year  MonthNumber  \
0     Thousand                55.4  1998            1
1     Thousand                31.5  1998            1
2     Thousand                23.9  1998            1
3     Thousand                96.2  1998            1
4     Thousand                55.4  1998            1
...        ...                 ...   ...          ...
1849  Thousand                22.5  2023            9
1850  Thousand                19.0  2023            9
1851  Thousand                74.2  2023            9
1852  Thousand                40.7  2023            9
1853  Thousand                33.5  2023            9

      Age Group_15 - 24 years  Age Group_25 - 74 years  Sex_Both sexes  
\
0                           1                        0               1

1                           1                        0               0

2                           1                        0               0

3                           0                        1               1

4                           0                        1               0

...                       ...                      ...             ...

1849                        1                        0               0

1850                        1                        0               0

1851                        0                        1               1

1852                        0                        1               0

1853                        0                        1               0


      Sex_Female  Sex_Male
0              0         0
1              0         1
2              1         0
3              0         0
4              0         1
...          ...       ...
1849           0         1
1850           1         0
1851           0         0
1852           0         1
1853           1         0
```

```
[1854 rows x 12 columns]
```

Now if we look at our dataset. we see that Unit and Unit.1 is not important for our analysis. so let's drop them. Removing unnecessary features make our analysis fast. these Unit and Unit.1 features are repeating some information.

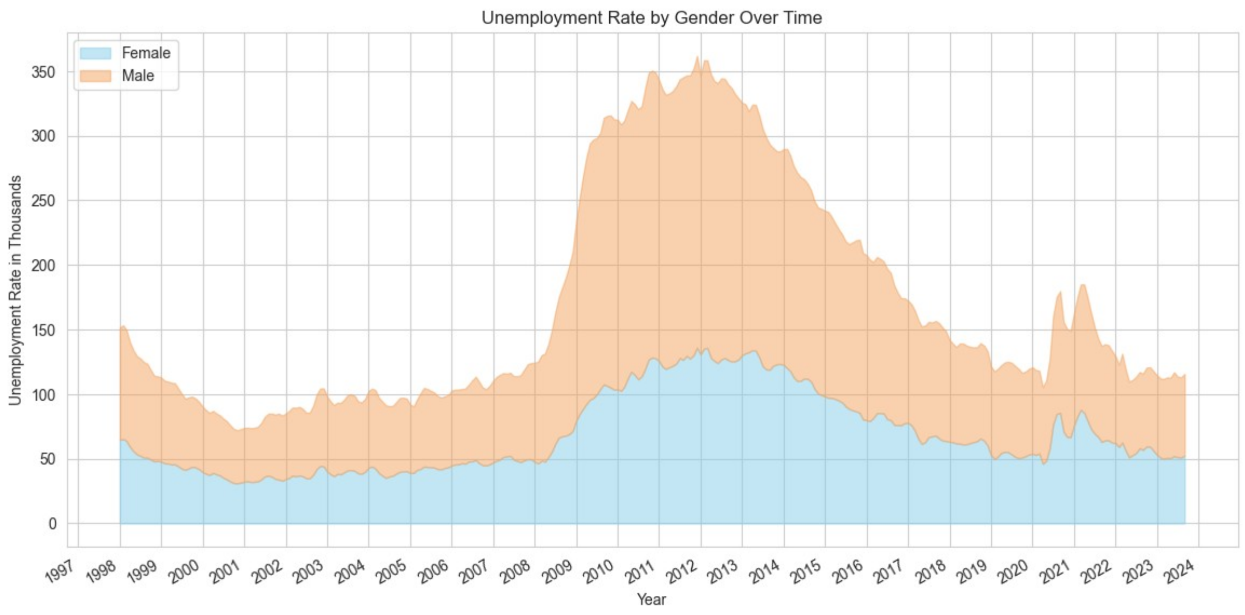
```
dropedUnitAndUnit1 =
encodedDataForAgeAndSex.drop(columns=['UNIT','UNIT.1'])
monthlyUnemploymentFilteredDF=dropedUnitAndUnit1
monthlyUnemploymentFilteredDF.head()
```

```
                                   Statistic Label  VALUE In Percentage
\
0  Seasonally Adjusted Monthly Unemployment Rate                  13.3

1  Seasonally Adjusted Monthly Unemployment Rate                  13.7

2  Seasonally Adjusted Monthly Unemployment Rate                  12.7

3  Seasonally Adjusted Monthly Unemployment Rate                   7.4

4  Seasonally Adjusted Monthly Unemployment Rate                   6.9


   VALUE In Thousands  Year  MonthNumber  Age Group_15 - 24 years  \
0                55.4  1998            1                        1
1                31.5  1998            1                        1
2                23.9  1998            1                        1
3                96.2  1998            1                        0
4                55.4  1998            1                        0

   Age Group_25 - 74 years  Sex_Both sexes  Sex_Female  Sex_Male
0                        0               1           0         0
1                        0               0           0         1
2                        0               0           1         0
3                        1               1           0         0
4                        1               0           0         1
```

Now let's create one more graph to show unemployment rate for Men and Women over time. First of all we will filter our data to get Men and women from it. After that we will sort the data by year and month and also by if it is of men and women. After that we will sum up all the unemployment for every group Then we will set the table so we can how things chnage by month and year and for each gender. We will label oour graph perfectly so it will be understanble by anyone that what is this graph about. We used 2 differewnt colors to differentiate between en and women data over time. we made it simple and easy to understand. Stacked like graphs are good when we add something to total accross time. and to show comparison of something over time. And with the help of this graph we can easily see which gender has more unemployemnt rate.

```
# Lets filter our data and get data for men and women
genderData =
monthlyUnemploymentFilteredDF[(monthlyUnemploymentFilteredDF['Sex_Fema
le'] == 1) | (monthlyUnemploymentFilteredDF['Sex_Male'] == 1)]

# Now lets organize data by year and month and gender and then we are
adding up unemployment number for every group
unemploymentCountByGender = genderData.groupby(['Year', 'MonthNumber',
'Sex_Male'])['VALUE In Thousands'].sum().reset_index()

# Now lets change the data so we can see it by year and month and for
seprate column for every gender
unemploymentCountByGenderPivot =
unemploymentCountByGender.pivot_table(index=['Year', 'MonthNumber'],
columns='Sex_Male', values='VALUE In Thousands')

# Rename columns for more clarity
unemploymentCountByGenderPivot.columns = ['Female', 'Male']

# Now lets reset index and get it ready to display
unemploymentCountByGenderPivot.reset_index(inplace=True)

# Now we will create a column called and combine yaer and month in a
dataframe
unemploymentCountByGenderPivot['Date'] =
pd.to_datetime(dict(year=unemploymentCountByGenderPivot.Year,
month=unemploymentCountByGenderPivot.MonthNumber, day=1))

# Now lets make our data in an order from early to latest
unemploymentCountByGenderPivot =
unemploymentCountByGenderPivot.sort_values('Date')

# Now set size of graph so it will be big enough to see

plt.figure(figsize=(14, 7))
```

```
# We plot the cumulative sum of Female unemployment first, then add
Male unemployment on top of it

# Now in teh following two lines we are filling the rae of
unemployment rate by  color, usiong 2 different colors for each gender
to differentiate between them
plt.fill_between(unemploymentCountByGenderPivot['Date'], 0,
unemploymentCountByGenderPivot['Female'], color="skyblue", alpha=0.5,
label='Female')
plt.fill_between(unemploymentCountByGenderPivot['Date'],
unemploymentCountByGenderPivot['Female'],
unemploymentCountByGenderPivot['Female'] +
unemploymentCountByGenderPivot['Male'], color="sandybrown", alpha=0.5,
label='Male')

# Now seeting the title, label, legend so it will be understanble for
anyone just by seeing at it that what is this graph about
plt.title('Unemployment Rate by Gender Over Time')
plt.xlabel('Year')
plt.ylabel('Unemployment Rate in Thousands')
plt.legend(loc='upper left')
plt.grid(True)

# Now making the date at the bottom and just dispalying year so make
it clear
plt.gca().xaxis.set_major_locator(plt.matplotlib.dates.YearLocator())
plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.DateFormatter
('%Y'))
plt.gcf().autofmt_xdate()

plt.show()
```



Question: Appropriate visualizations must be used to engender insight into the dataset and to illustrate your final insights gained in your analysis. [0-20]

All design and implementation of your visualizations must be justified and detailed in full. [0-30]

As you can look into the notebook that we have done the EDA and feature engineering of our data. From loading our libraries to feature enginnerign every step is explained. I expained before doing every stepp that what we are doing and agter implementing that what we gained after doing this. I also put comments in my code.

I used the perfect graphs for each problem, for example when we plot the unemployment rate by age group we used teh boxplot insead of other graohs because we know that boxplots are good options in such cases where we show the distribution and spread of data.

We used the line graph to shoe unemployemnt rate over time because as we know line graph can easily show the good and bad times about our data and anyone can easily see that just by looking at it.

When we shoed the distribution of our data we used histogram to plot it and we know it is primary tool to understand the numerical data.

While doing the EDA I also visualized through appropriote graphs. and explained that why we used them.

Statistics for Data Analytics

Question: Summarise your dataset clearly, using relevant descriptive statistics and appropriate plots. These should be carefully motivated and justified, and clearly presented. You should critically analyse your findings, in addition to including the necessary Python code, output and plots in the report. You are required to plot at least three graphs. [0-35]

Lets start by looking into the dataset taht what we have in it and about data types or any null values

```
monthlyUnemploymentFilteredDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1854 entries, 0 to 1853
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Statistic Label        1854 non-null   object 
 1   VALUE In Percentage    1854 non-null   float64
 2   VALUE In Thousands     1854 non-null   float64
 3   Year                   1854 non-null   int64  
 4   MonthNumber            1854 non-null   int64  
 5   Age Group_15 - 24 years  1854 non-null   uint8  
 6   Age Group_25 - 74 years  1854 non-null   uint8  
 7   Sex_Both sexes         1854 non-null   uint8  
 8   Sex_Female             1854 non-null   uint8  
 9   Sex_Male               1854 non-null   uint8  
dtypes: float64(2), int64(2), object(1), uint8(5)
memory usage: 81.6+ KB
```

This above inforation shows that we have total 1854 rows and 10 ccolumns in total. First Of all we got the 'Statistic Label' column which has no null value and is of string type. This olumn is woking like a label and telling us we heve unemployment data. The second column 'VALUE In Percentage' has unemployment rate in percentage. there are no null values in the dataset and the datatype of it is float then the next column 'VALUE In Thousands' shows the number of unemployed people in thousands. and it's not null as well. Then Age Group_15 - 24 years and Age Group_25 - 74 years are binary column and accept 0 and 1 as value and are all not null. Sex_Both sexes , Sex_Female AND Sex_Male are binary column too indicating gender category. Year and monthNuber are extracted from month column and are of int type and all are not null.

Now let's calculate the mean, median, variance, and standard deviation on whole dataset. First We will calculate the basic measures like Mean, median, variance and standard deviation in percentage and in thousands as well.

```
# Basic measures for whole dataset

basicMeasuresForWholeData ={
    'Mean Of VALUE In Percentage' :
monthlyUnemploymentFilteredDF['VALUE In Percentage'].mean(),
    'Median Of VALUE In   Percentage' :
monthlyUnemploymentFilteredDF['VALUE In Percentage'].median(),
    'Variance Of VALUE In   Percentage' :
monthlyUnemploymentFilteredDF['VALUE In Percentage'].var(),
    'Standard Deviation Of VALUE In  Percentage' :
monthlyUnemploymentFilteredDF['VALUE In Percentage'].std(),

    'Mean Of VALUE In Thousands' :
monthlyUnemploymentFilteredDF['VALUE In Thousands'].mean(),
    'Median Of VALUE In  Thousands' :
monthlyUnemploymentFilteredDF['VALUE In Thousands'].median(),
    'Variance Of VALUE In  Thousands' :
monthlyUnemploymentFilteredDF['VALUE In Thousands'].var(),
    'Standard Deviation Of VALUE In  Thousands' :
monthlyUnemploymentFilteredDF['VALUE In Thousands'].std()
}
basicMeasureTableDF=pd.DataFrame(basicMeasuresForWholeData, index
=['Value']).T
display(basicMeasureTableDF);
```

```
                                                Value
Mean Of VALUE In Percentage                 10.680529
Median Of VALUE In   Percentage              8.800000
Variance Of VALUE In   Percentage           56.840187
Standard Deviation Of VALUE In  Percentage   7.539243
```

```
Mean Of VALUE In Thousands                           56.665696
Median Of VALUE In  Thousands                        39.000000
Variance Of VALUE In  Thousands                    2356.237840
Standard Deviation Of VALUE In  Thousands            48.541094
```

Now let's calculate the mean for just males and females. By breaking down the data by genders will help us understand the unemployment rate by genders. This way we can get deeper understanding of our dataset. It will help us understand if certain genders are affected by uneployment.

```
# Let's first calculate Mean for just Males

# First we will filter the data to only get rows where gender is Male
maleMean=monthlyUnemploymentFilteredDF[monthlyUnemploymentFilteredDF['
Sex_Male']==1]['VALUE In Percentage']

# Now let's find the descriptives for example mean,median, quantiles
for males unemployment rate in percentage
maleMean = maleMean.describe()

maleMean['variance'] = maleMean.var()

print("Measures For Males:\n",maleMean)

Measures For Males:
 count          618.000000
mean            11.948382
std              9.083101
min              2.900000
25%              4.700000
50%              9.600000
75%             14.975000
max             39.300000
variance     45848.319195
Name: VALUE In Percentage, dtype: float64
```

With the above analysis we disovered that the unemployment rate for males is 11.95 percent (on average) that means around 11.95 percent of males are unemployed. The median is 9.6% which means in half of our data points the unemployment rate for males is under 9.6%. With the help of this information we get to know that in some time period unemployment rate for males migh be considerably high.

The variance is 82.5 and Standard deviation is 9.08 %. this tells us that the unemployment rate for males vary in our dataset. This high variation could be because of different reasons. Well the lowest unemployment rate is 2.9 percent and the highest is 39.3 percent.

let's talk about the quantiles, 25 percent of data points have unemployment rate below 4.7 for males. and 75percent of the data records rate below 14.98 percent. This gives us interquantile range and emphasizes entral clustring of most of our data.

```
# Let's use the same way to calculate Mean for  Females

# First we will filter the data to only get rows where gender is Male
femaleMean=monthlyUnemploymentFilteredDF[monthlyUnemploymentFilteredDF
['Sex_Female']==1]['VALUE In Percentage']

# Now let's find the descriptives for example mean,median, quantiles
for males unemployment rate in percentage
femaleMean = femaleMean.describe()

femaleMean['variance'] = femaleMean.var()

print("Measures For Females:\n",femaleMean)
```

```
Measures For Females:
 count        618.000000
mean           9.359061
std            5.564753
min            2.900000
25%            4.700000
50%            7.900000
75%           11.475000
max           24.600000
variance   46329.282255
Name: VALUE In Percentage, dtype: float64
```

The results above for Females shows a slightly different image. Average unemployment rate for females is around 9.36 percent. which is lower than males. so females faed lower unemployment rate in our dataset. The median is 7.9 for females whih means half of our data points record an unemployment rate below this number.

Variance and standard deviation is 30.97 and 5.56 percent. if we compare this with males these numbers are lower which means that female unemployment rates are less variable and more consistant. The minimum unemployment rate for females is same of males. but the maximum number is lower at 24.6 percent. This shows that feales have lower unemployment rates than males.

Let's calculate the measures for each Age Group

```
# Let's use the same way to calculate Mean for  Females

# First we will filter the data to only get rows where age group is 15
to 24
age15To24Measures=monthlyUnemploymentFilteredDF[monthlyUnemploymentFil
teredDF['Age Group_15 - 24 years']==1]['VALUE In Percentage']

# Now let's find the descriptives for example mean,median, quantiles
for 15 to 24 age group unemployment rate in percentage
age15To24Measures = age15To24Measures.describe()

age15To24Measures['variance'] = age15To24Measures.var()

print("Measures For Age Group_15 - 24 years:\n",age15To24Measures)
```

```
Measures For Age Group_15 - 24 years:
 count          927.000000
mean            15.038080
std              8.015164
min              5.900000
25%              8.900000
50%             11.900000
75%             19.450000
max             39.300000
variance    103966.348307
Name: VALUE In Percentage, dtype: float64
```

```
# Let's use the same way to calculate Mean for  Females

# First we will filter the data to only get rows where age group is 15
to 24
age25To74Measures=monthlyUnemploymentFilteredDF[monthlyUnemploymentFil
teredDF['Age Group_25 - 74 years']==1]['VALUE In Percentage']

# Now let's find the descriptives for example mean,median, quantiles
for 15 to 24 age group unemployment rate in percentage
```

```
age25To74Measures = age25To74Measures.describe()

age25To74Measures['variance'] = age25To74Measures.var()

print("Measures For Age Group_25 - 74 years:\n",age25To74Measures)
```

```
Measures For Age Group_25 - 74 years:
 count          927.000000
mean             6.322977
std              3.388419
min              2.900000
25%              3.800000
50%              4.700000
75%              8.350000
max             15.600000
variance    105946.509165
Name: VALUE In Percentage, dtype: float64
```

In above 2 cells we calculated the measures by age group. This way we can find if unemployment rates are different for younger and older people. And teh numbers showed us that the younger people faced more challanges than older people. Well this could be because of many reasons. For example younger people have less work experience, maybe the skills they possess are not in demand very much,transition from studies to work , or it could be because of economic factors.

For Age Group 15 - 24 years the measures says that the average unemployment rate is 15.04 pecent. which is high number especially when we check the middle value which is 11.9 percent.

Then the Variance and standard deviation is 64.24 and 8.02 percent, these numbers learlly show the fluctuation in unemployment rate. Then the 25 and 75 % of data points of this age group have unemployment below 8.9 and 19.45 respectively. This helps us to know that where most of the unemployment rates falls and show this age group often faes job challenges.

For Age Group 25 - 74 years the numbers are different. The average unemployment rate is 6.32 percent which is lower than younger people group. Median is lower too at 4.7 percent that means half of the data points record unemployment rate below this number.

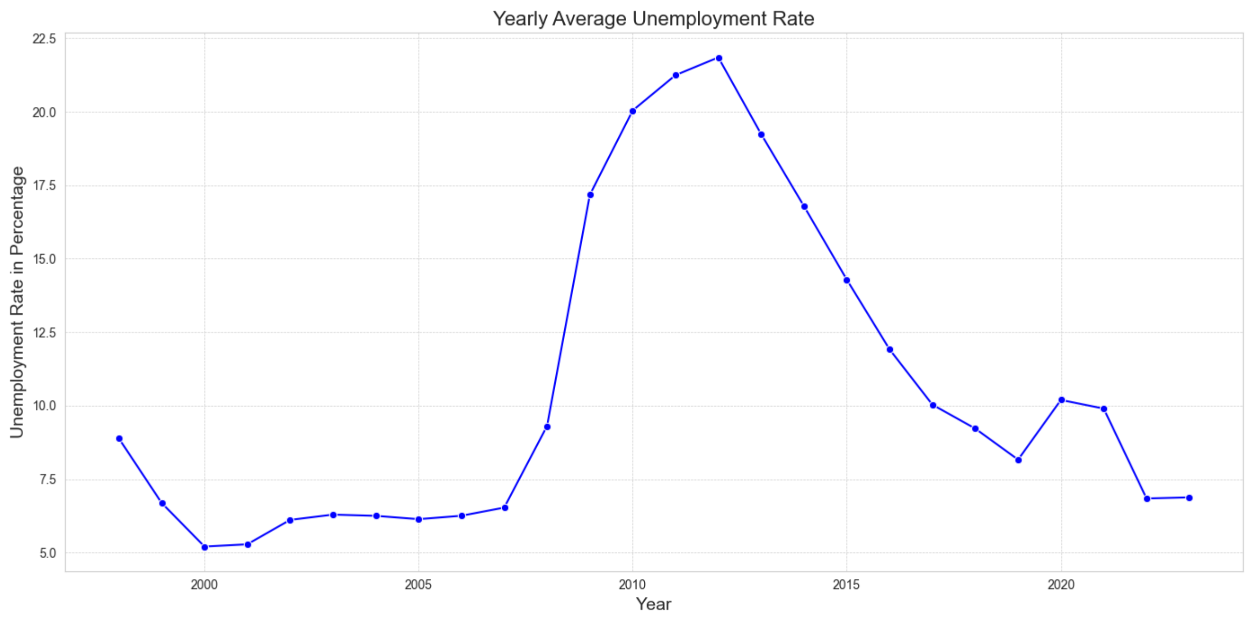
the Variance and standard deviation is 11.48 and 3.39 and that is also lower than younger group. This means that this age group had fewer fluctuatuations and unemployment arte is stable as compared to younger group.

The minimum returned 2.9 and maximum is 15.6 for this age group . This means 25 - 74 years dosen't face the same unemployment issues. The 25 and 75 % of quantiles are 3.8% and 8.35 %. this means that the unemployment rate dosen't vary too uh for this age group.

We have done EDA and statistical analysis on our data. we looked for null values. and total count of our data. also checked for data types and then Calculated Mean, Median, Variance, Standard Deviation, Minium, Maximum, and Quantiles of our whole dataset and of groups as well. Now we knwo what our data says about unemployment rate for each group. Now let's plot graphs and see our data visually.

Let's Visualize the yearly average unemployment Rate. With the help of this graph we can identify the frequency of diffrent unempployment rates in our dataset. We can use the histogram to show the distribution of unemployment rate. Histogram is good to understand the distribution of continous data.

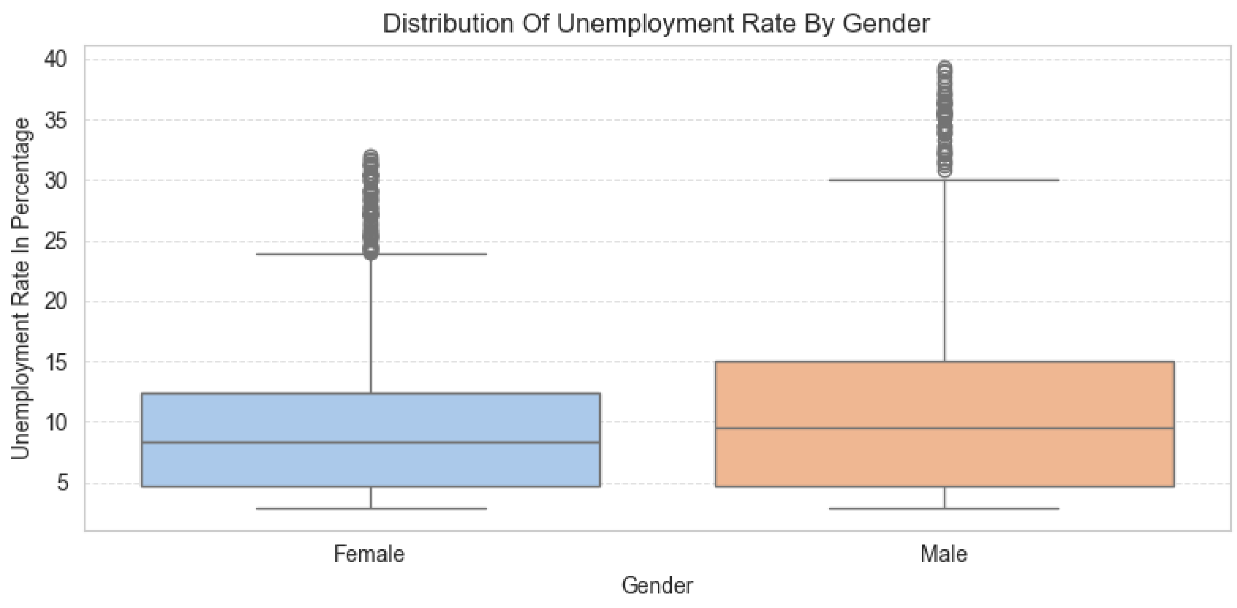
```
# To plot yearly unemployment rate, we need to find the mean and group the data by year

yearlyUnemploymedntRateFiltered=monthlyUnemploymentFilteredDF.groupby('Year')['VALUE In Percentage'].mean().reset_index()
yearlyUnemploymedntRateFiltered.columns=['Year','Average Unemployment Rate']

# we grouped our data so now let's plot graph
plt.figure(figsize=(14,7))
sbn.lineplot(data=yearlyUnemploymedntRateFiltered,x='Year',y='Average Unemployment Rate',color='blue',marker='o')
plt.title("Yearly Average Unemployment Rate", fontsize=16)
plt.xlabel('Year',fontsize=14)
plt.ylabel('Unemployment Rate in Percentage',fontsize=14)
plt.grid(True,which='both',linestyle='--',linewidth=0.5)
plt.tight_layout()
plt.show()
```



This graph shows the change in unemployment rate in diffrent years. With the help of this graoh we an see which year has the highest unemployment rate. We can see a spike in uneployment rate then there is deline and then again increase in the unemployment.

Lets display second graph to show unemployment rate by gender

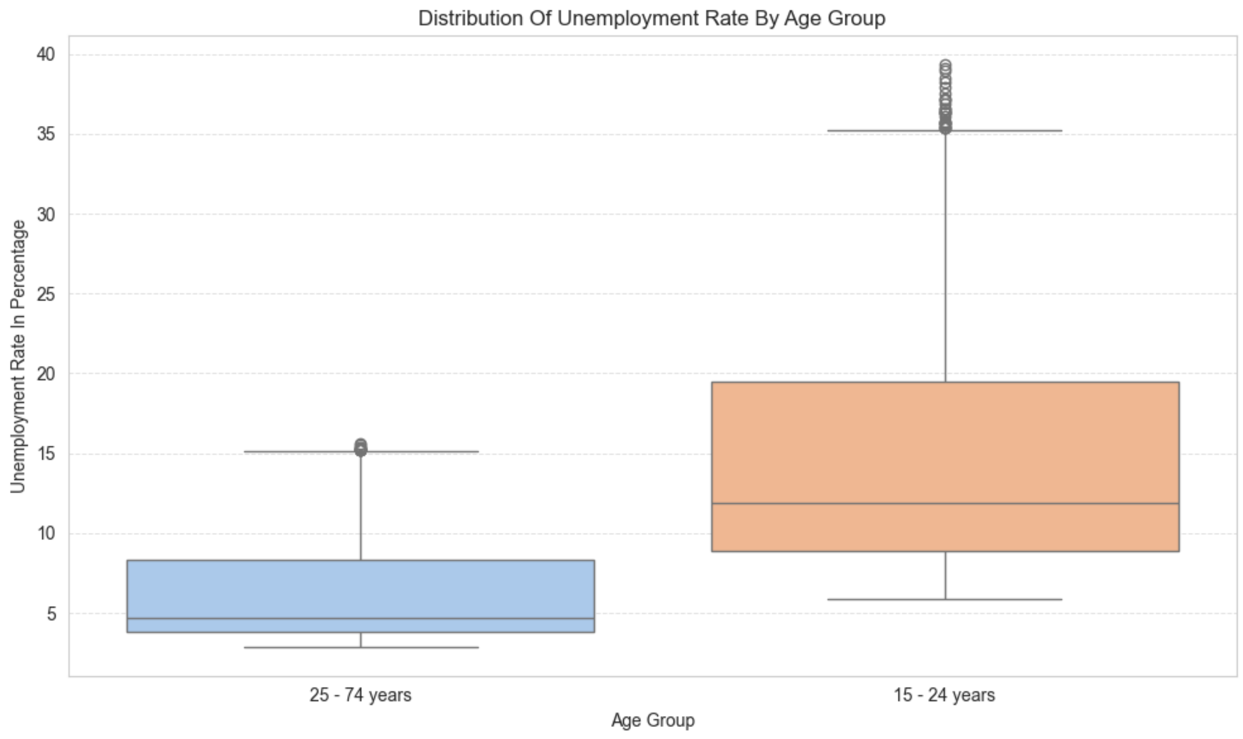
```
plt.figure(figsize=(8,4))
sbn.boxplot(data=monthlyUnemploymentFilteredDF,x='Sex_Male',y='VALUE In Percentage',palette='pastel')
plt.title('Distribution Of Unemployment Rate By Gender')
plt.xlabel('Gender')
plt.ylabel('Unemployment Rate In Percentage')
plt.grid(True,axis='y',linestyle='--',linewidth=0.7,alpha=0.6)
plt.xticks([0,1],['Female','Male'])
plt.tight_layout()
plt.show()
```



As we already discussed while calculating mean for group data that the unemployment rate for males is slightly high than women. We acn see that with the help of graph that the male unemployment rate is higher than wemen. Boxplot give us look that how data is spread out. and with the help of lines and boxes we can see that where most of the data falls and if there are any differenecs.

Now let's plot another graph of unemployment rate by Age Groups and justify our Measures. We will be using the histogram again to display teh data.

```
plt.figure(figsize=(10,6))
sbn.boxplot(data=monthlyUnemploymentFilteredDF,x='Age Group_15 - 24
years',y='VALUE In Percentage',palette='pastel')
plt.title('Distribution Of Unemployment Rate By Age Group')
plt.xlabel('Age Group')
plt.ylabel('Unemployment Rate In Percentage')
plt.grid(True,axis='y',linestyle='--',linewidth=0.7,alpha=0.6)
plt.xticks([0,1],['25 - 74 years', '15 - 24 years'])
plt.tight_layout()
plt.show()
```



As we can see that the younger age group has higher median unemployment rate as compared to older age group. And their spread is wide too. This means that younger face more challanges than older people. This could be because of many reasons as i explained before while calculating measures.with these charts and numbers we can see unemployment for different age groups.

Question: Use two discrete distributions (Binomial and/or Poisson) in order to explain/identify some information about your dataset. You must explain your reasoning and the techniques you have used. Visualise your data and explain what happens with the large samples in these cases. You must work with Python and your mathematical reasoning must be documented in your report

We have been asked to if we can explain our data with some shapes. The two shapes or patterns are mentioned in the question 1) Binomial and Poisson.

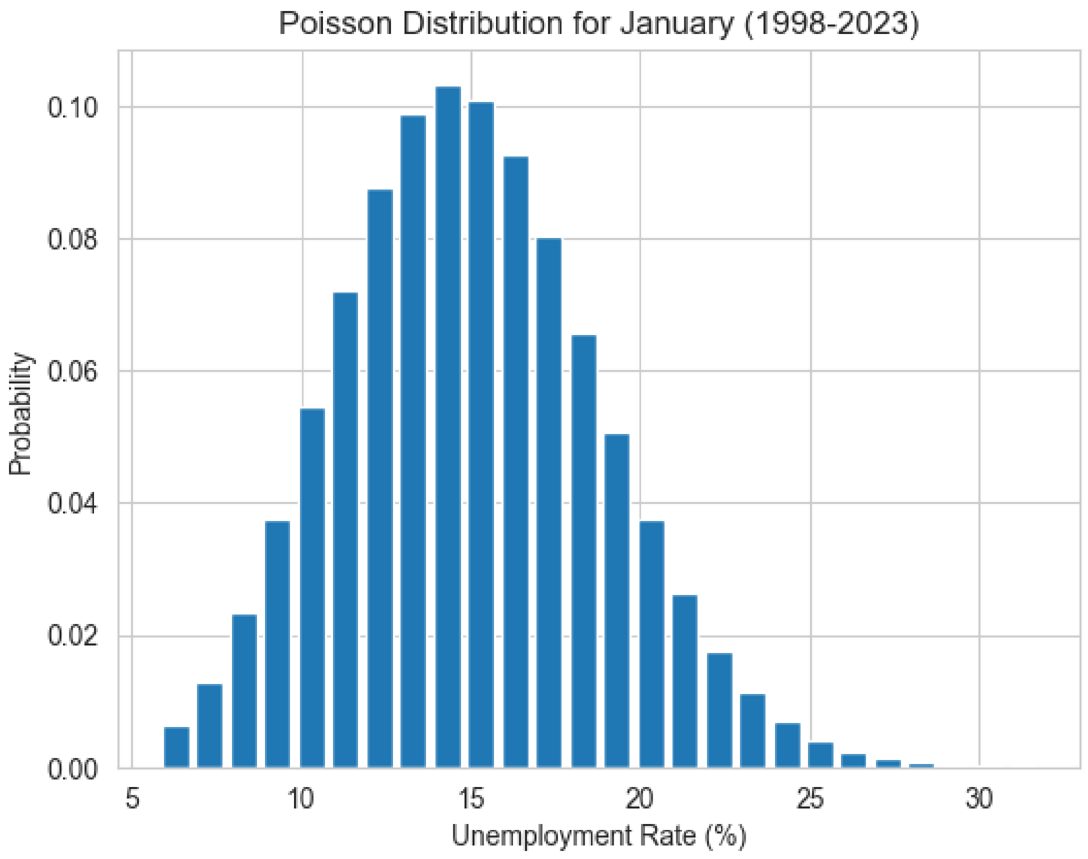
Bionomial distribution describes the number of successes in fixed number of independent Bernoulli trials. Bernoulli trial is an experiment with just 2 outcomes, success or failure. We can understand with this easy example that, on flipping a coin. if we someone gets head so he will get the job and if you get tails you won't get the job. If we flip coin for many people so binomial distribution can tell us that how many people will end up without a job.

Poisson distribution tells the number of events in a fixed interval of time. Let's take an exampe to understand this as well. Imagine a big ity in which someone losses a job now and then. Poisson distribution can help us guess how many people might loss their jobs in a month.

As we know that the poisson is perfect when count that how many times something happens in a certain time.for example if there is raining and you see rain drops falling on window. it is hard to predict that when rain drop will fall on window but we can count them over time. same, we can't predict when someone will get unemployed but we can count that how many people do get unemployed each month. A s our data is of a specific time so let's use poisson distrubtion to identify some information about our datase.

```python
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 1) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for January (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
```

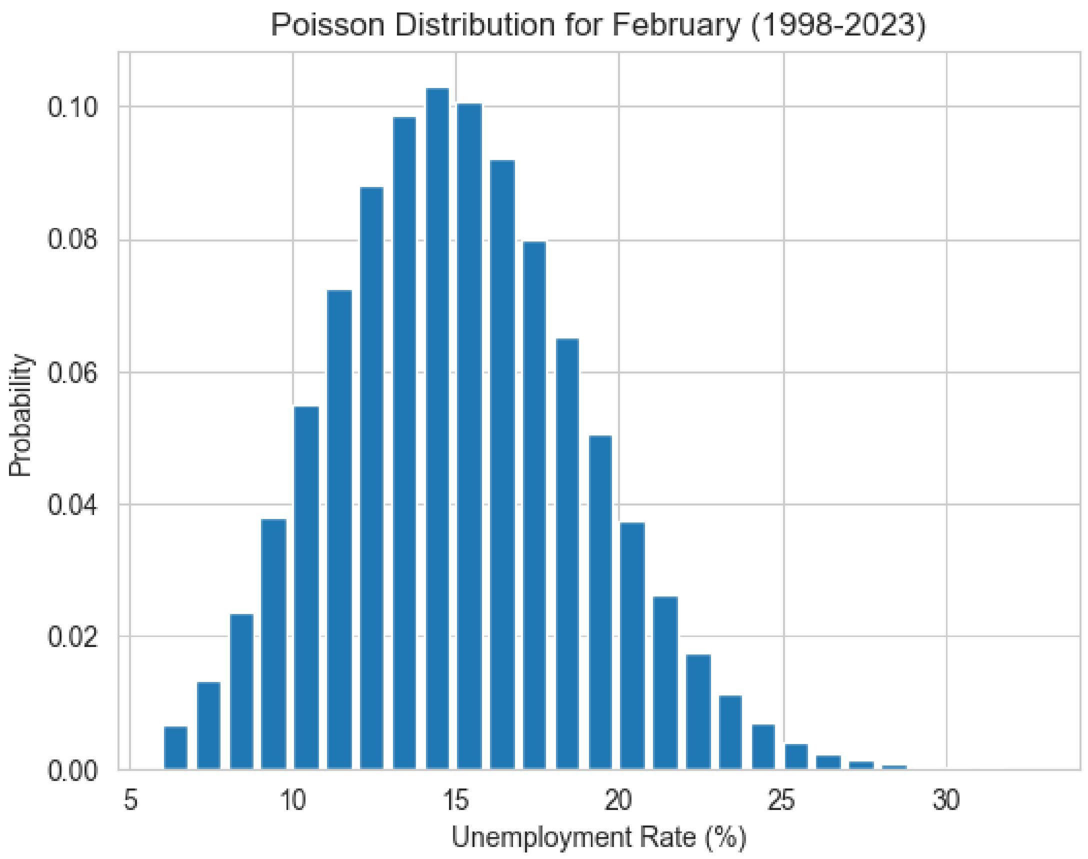
```
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()

   X (Number of Occurrences)  Probability
0                        6.3     0.006315
1                        7.3     0.012942
2                        8.3     0.023329
3                        9.3     0.037531
4                       10.3     0.054517
```



```

# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 2) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployeemnt rate from our filtered
```

```
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for February (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

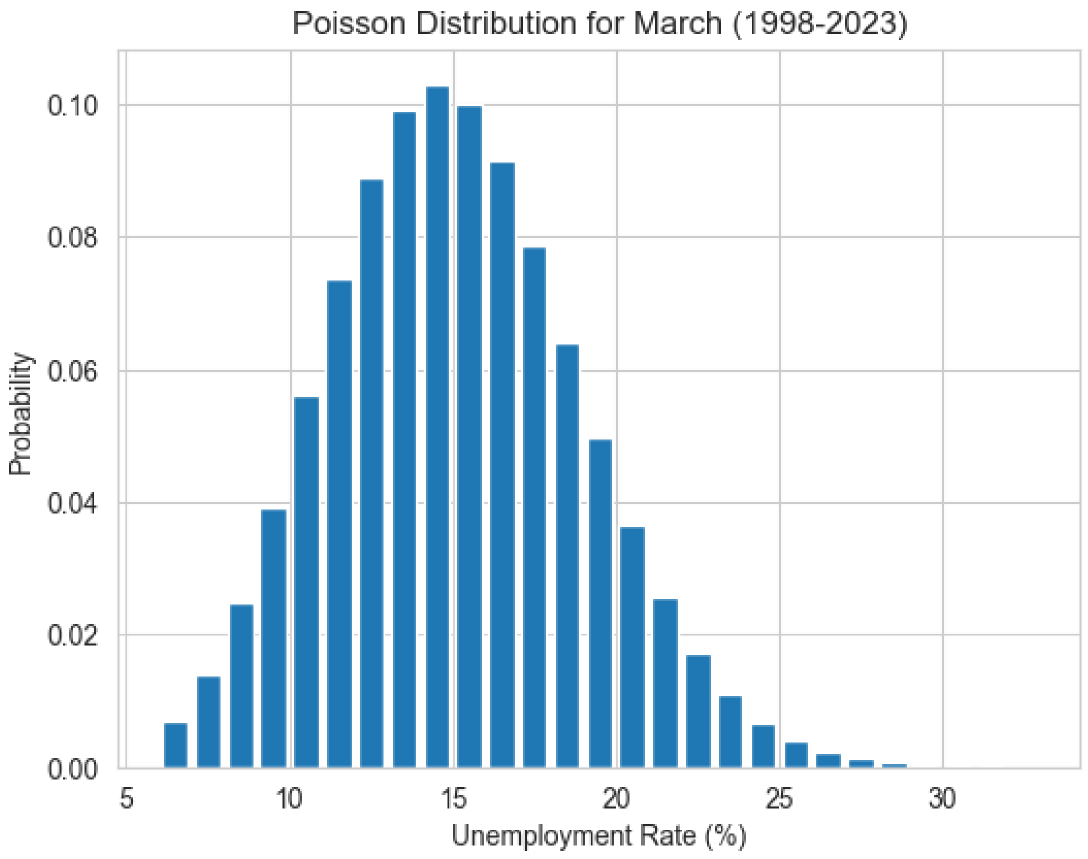
```
   X (Number of Occurrences)  Probability
0                        6.4     0.006533
1                        7.4     0.013276
2                        8.4     0.023767
3                        9.4     0.038024
4                       10.4     0.054983
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 3) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for March (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

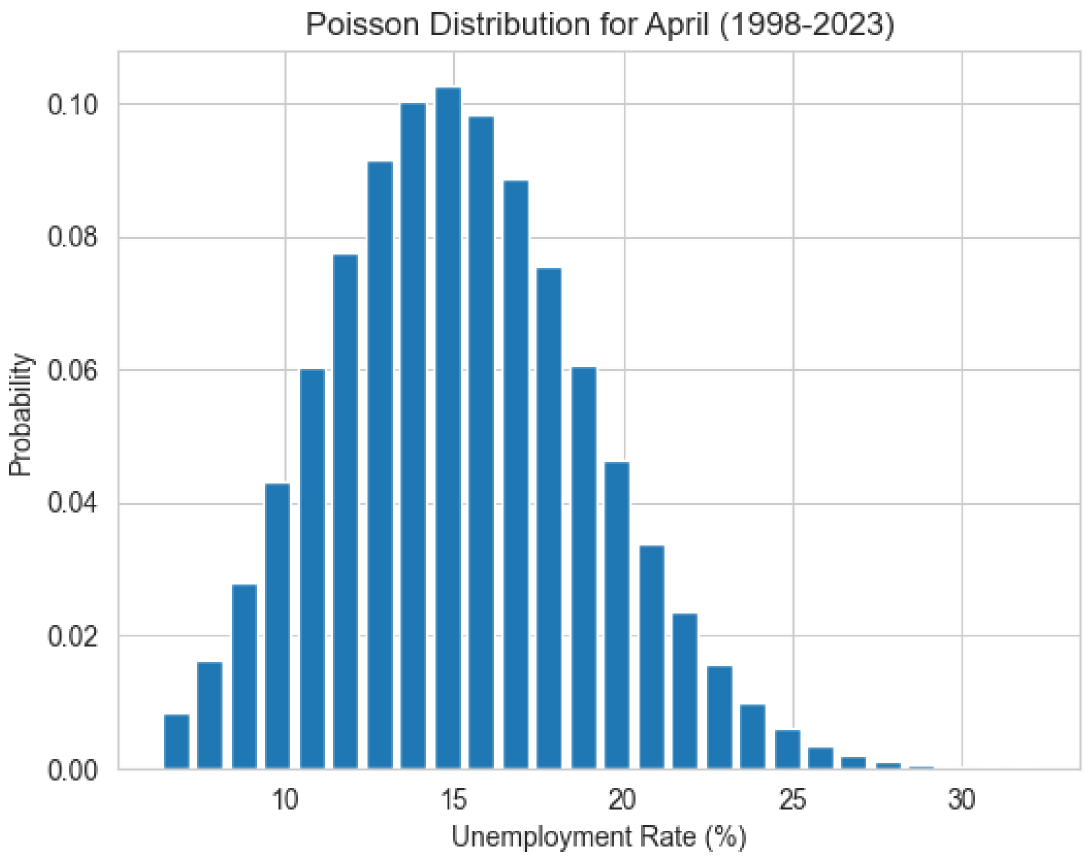
```
   X (Number of Occurrences)  Probability
0                        6.5     0.006934
1                        7.5     0.013932
2                        8.5     0.024699
```

```
3                              9.5     0.039178
4                             10.5     0.056227
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 4) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployeemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for April (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

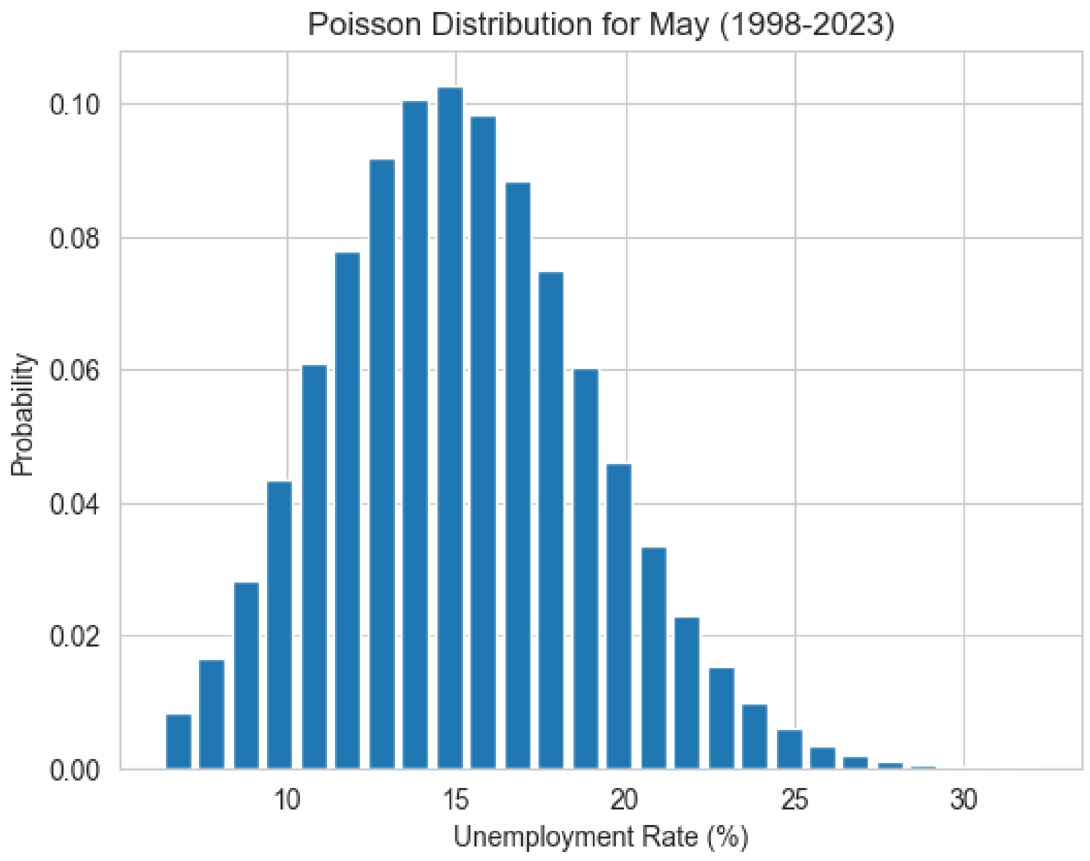
```
   X (Number of Occurrences)  Probability
0                        6.8     0.008364
1                        7.8     0.016229
2                        8.8     0.027911
3                        9.8     0.043104
4                       10.8     0.060404
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 5) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for May (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

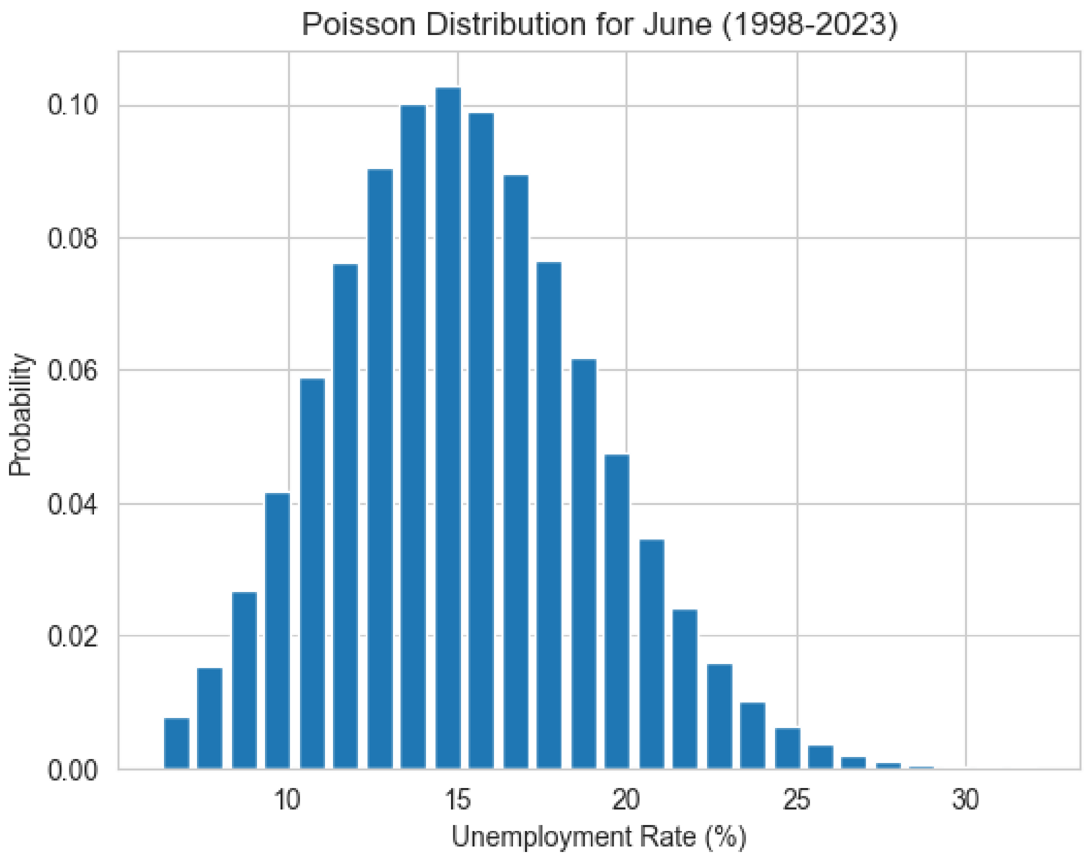
```
   X (Number of Occurrences)  Probability
0                        6.8     0.008507
1                        7.8     0.016472
2                        8.8     0.028272
3                        9.8     0.043573
4                       10.8     0.060937
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 6) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for June (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

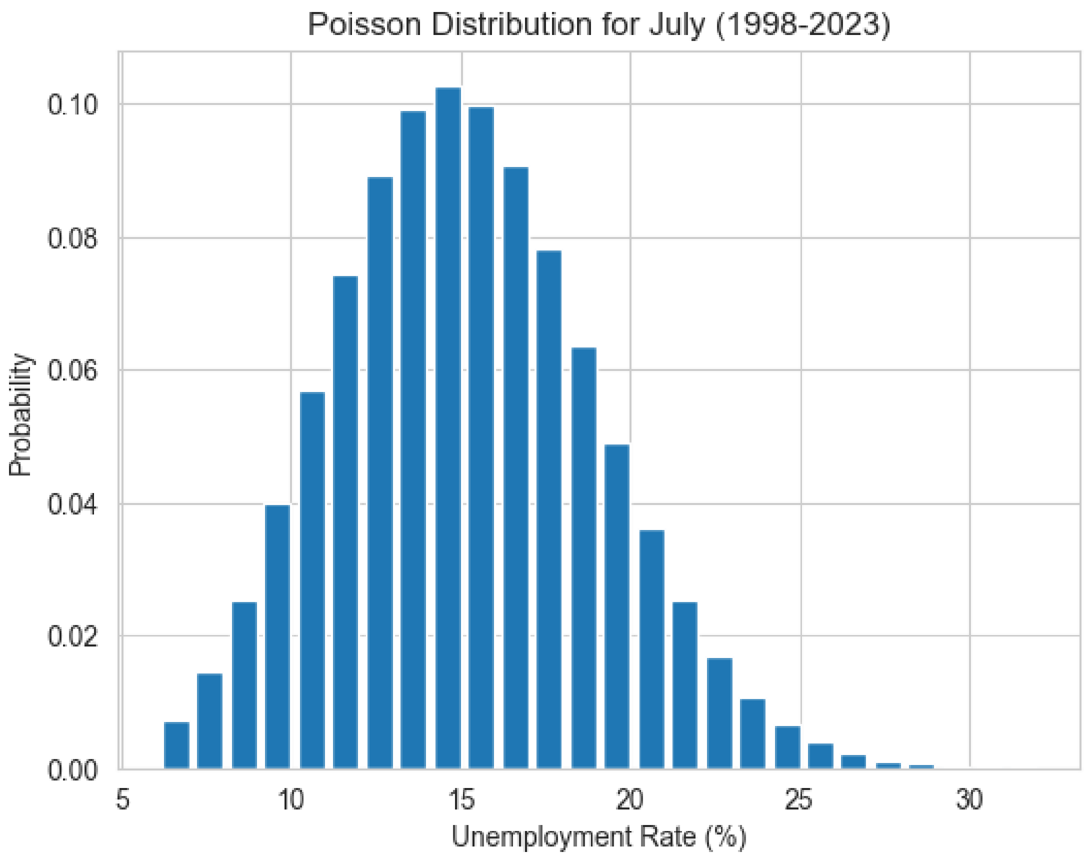
```
   X (Number of Occurrences)  Probability
0                        6.7     0.007838
1                        7.7     0.015390
2                        8.7     0.026745
3                        9.7     0.041687
4                       10.7     0.058905
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 7) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for July (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

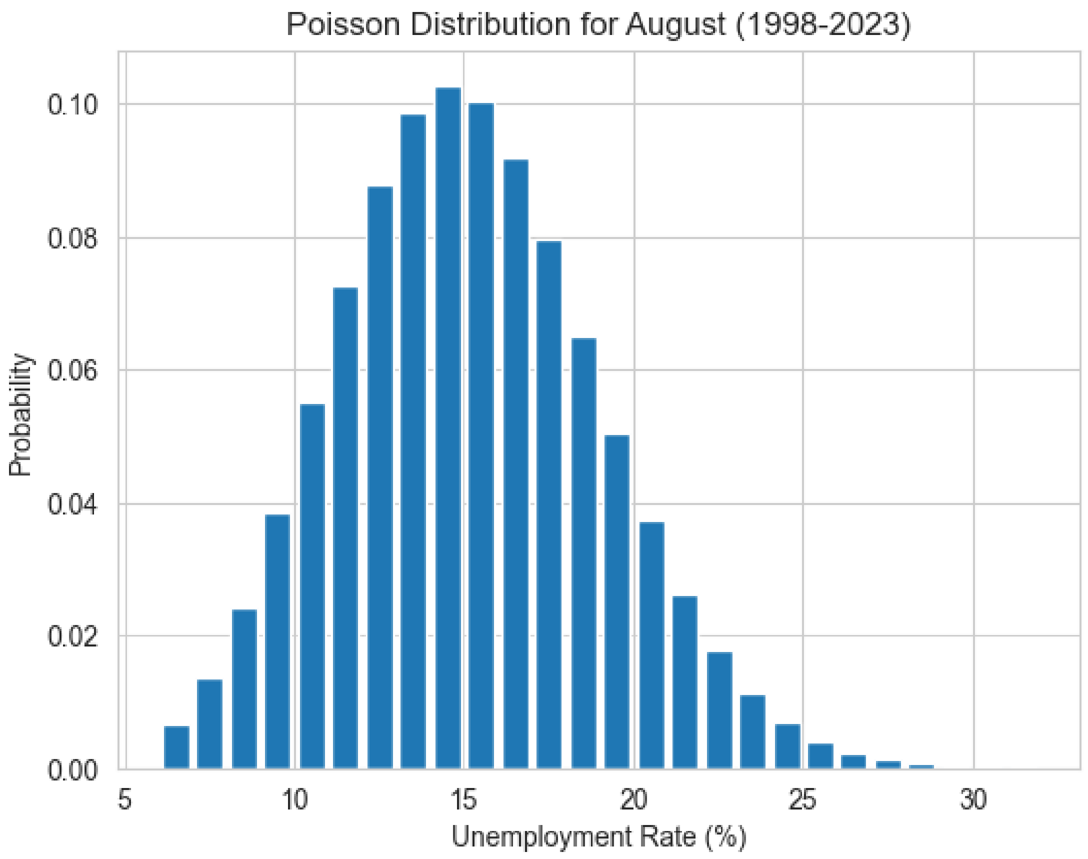
```
   X (Number of Occurrences)  Probability
0                        6.6     0.007226
1                        7.6     0.014386
2                        8.6     0.025310
3                        9.6     0.039892
4                       10.6     0.056943
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 8) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for August (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

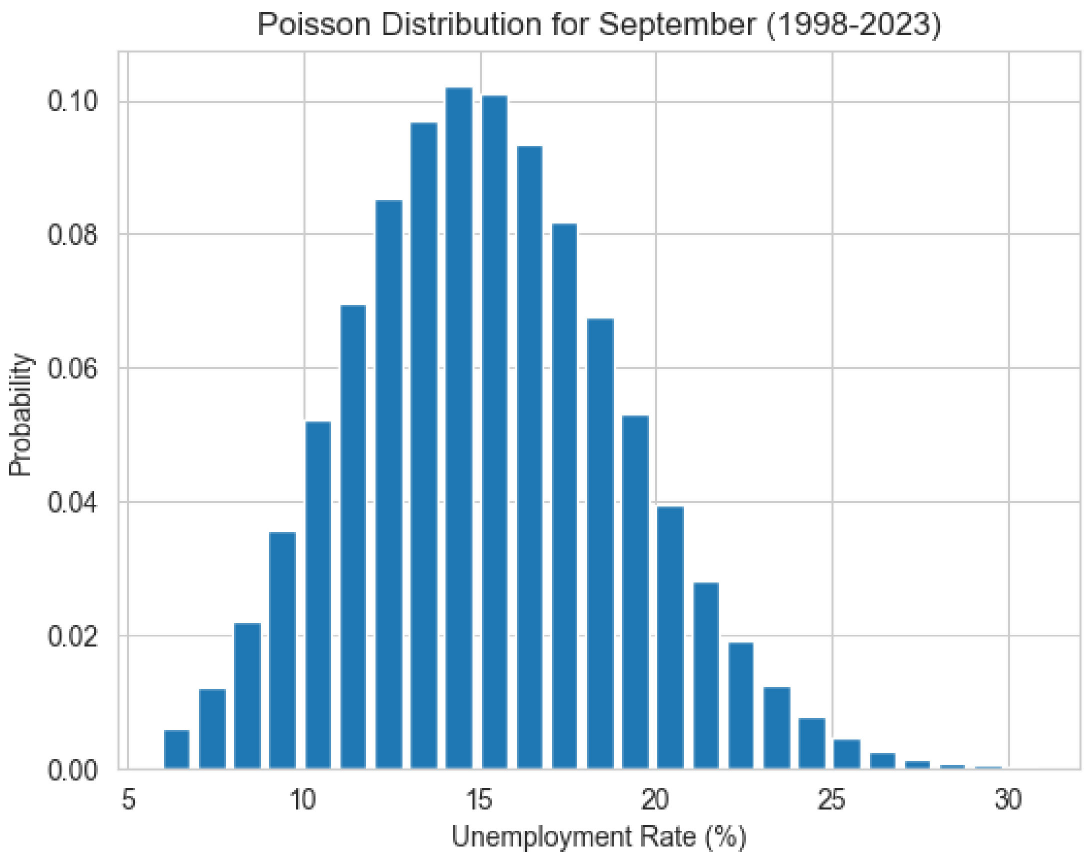
```
   X (Number of Occurrences)  Probability
0                        6.5     0.006710
1                        7.5     0.013533
2                        8.5     0.024084
3                        9.5     0.038349
4                       10.5     0.055248
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 9) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for September (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

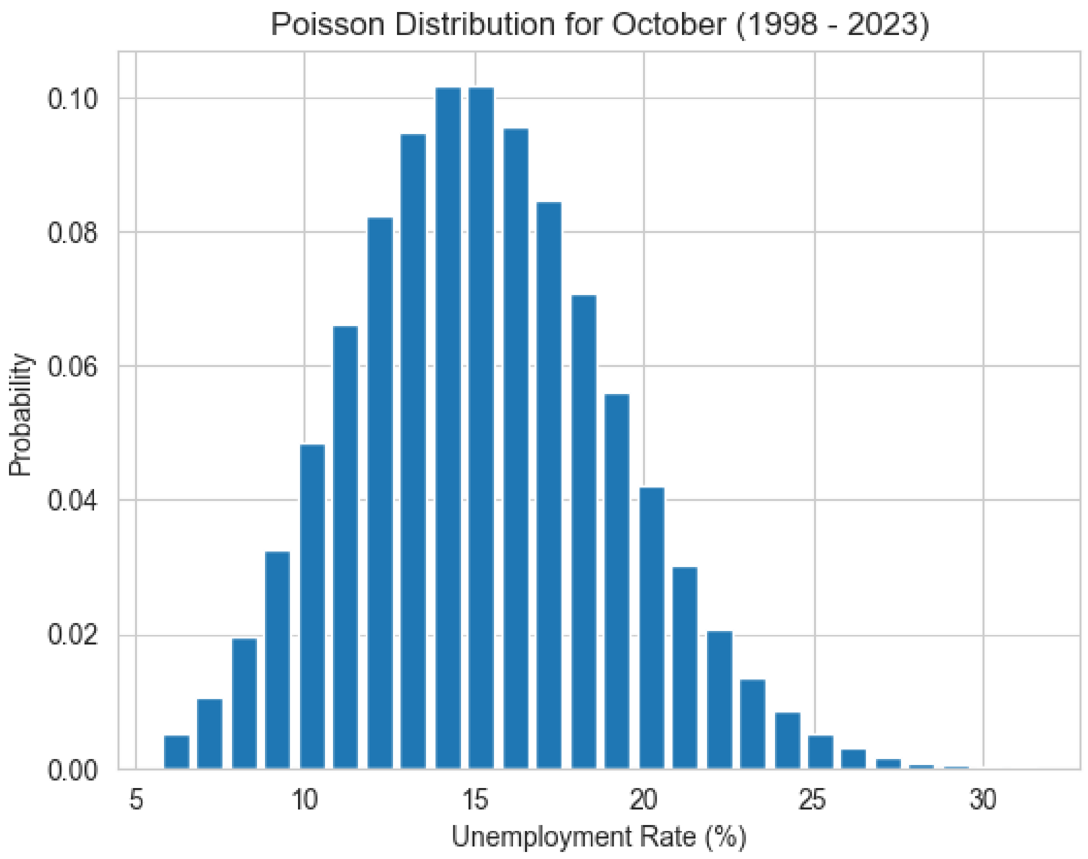
```
   X (Number of Occurrences)  Probability
0                        6.4     0.005938
1                        7.4     0.012201
2                        8.4     0.022083
3                        9.4     0.035717
4                       10.4     0.052216
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 10) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for October (1998 - 2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

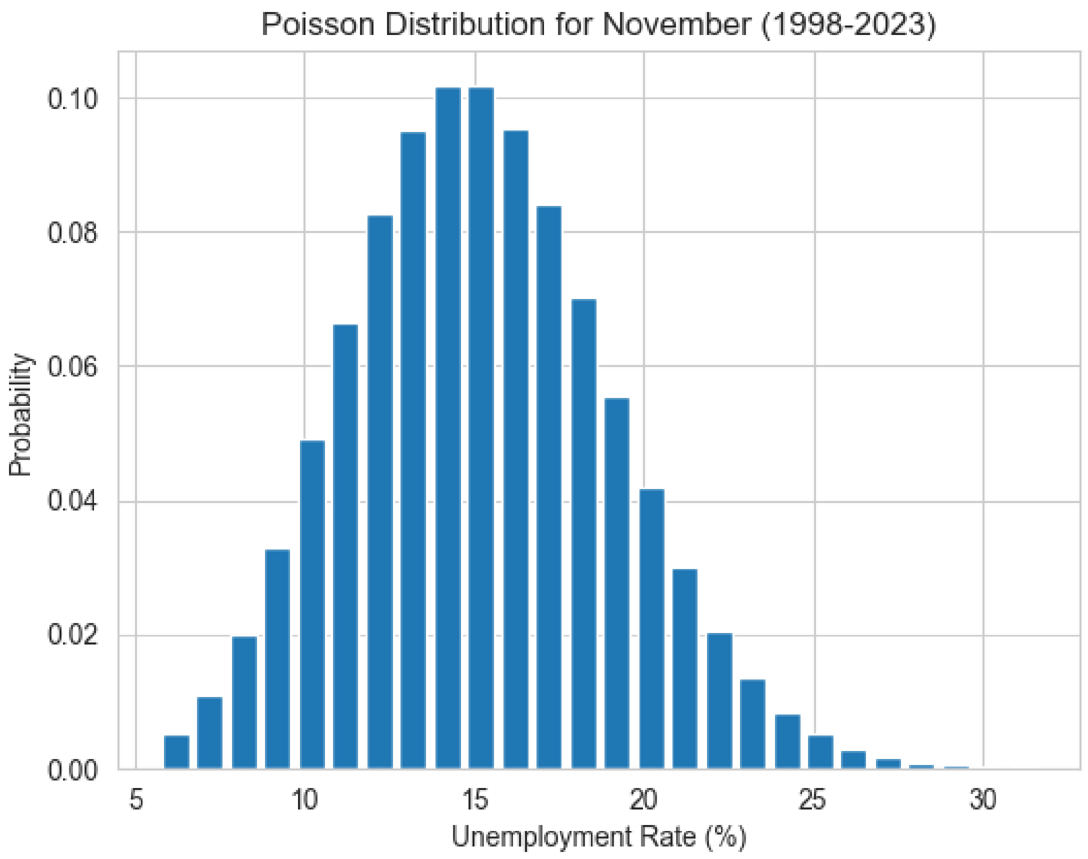
```
   X (Number of Occurrences)  Probability
0                        6.2     0.005020
1                        7.2     0.010609
2                        8.2     0.019687
3                        9.2     0.032560
4                       10.2     0.048572
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 11) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for November (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

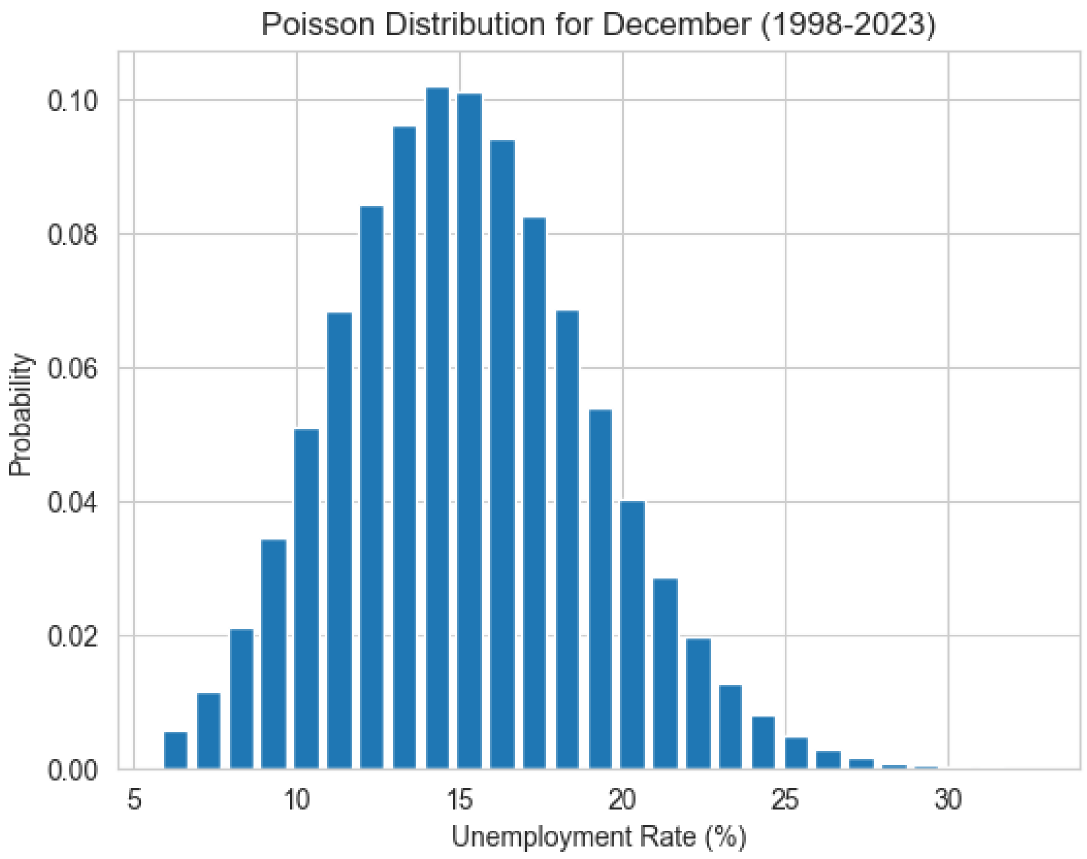
```
   X (Number of Occurrences)  Probability
0                        6.2     0.005116
1                        7.2     0.010789
2                        8.2     0.019979
3                        9.2     0.032974
4                       10.2     0.049086
```



```
# First of all lets filter our data by age group and gender
januaryDatForPoissonDist = monthlyUnemploymentFilteredDF[
    (monthlyUnemploymentFilteredDF['Age Group_15 - 24 years'] == 1) &
    (monthlyUnemploymentFilteredDF['Sex_Both sexes'] == 1) &
    (monthlyUnemploymentFilteredDF['MonthNumber'] == 12) &
    (monthlyUnemploymentFilteredDF['Year'] >= 1998) &
    (monthlyUnemploymentFilteredDF['Year'] <= 2023)
]

# Now lets find lowest and highest unemployemnt rate from our filtered
data
minValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].min()
maxValueOfFilteredData = januaryDatForPoissonDist['VALUE In
Percentage'].max()

# Now lets adjust teh unemployment rate to a scale form 0 to 1 . this
is a good practice to do for statistical models
unemploymentPercentage = januaryDatForPoissonDist['VALUE In
Percentage'] / 100

# now lets calculate average unemployemnt rate whcih is lambda for our
```

```
poisson distribution
lambdaValue = np.mean(unemploymentPercentage) * 100

# lets set teh range of unemployment to predict probabilities from
lowest to highest value
valuesOfX = np.arange(minValueOfFilteredData, maxValueOfFilteredData +
1)

# now lets sue the poisson distributon formula by calculating the
chance of unemployemnt in our range
poissonModelValues = [poisson.pmf(value, lambdaValue) for value in
valuesOfX]

# lets craete a table to display predicted probabilities and
unemployemnt rate
predictionTable = pd.DataFrame({
    'X (Number of Occurrences)': valuesOfX,
    'Probability': poissonModelValues
})

# Now lets display it
print(predictionTable.head())

# And now lets dipslay the graph to get more understanging of it
plt.bar(predictionTable['X (Number of Occurrences)'],
predictionTable['Probability'])
plt.xlabel('Unemployment Rate (%)')
plt.ylabel('Probability')
plt.title('Poisson Distribution for December (1998-2023)')
plt.grid(True)

# Now lets uise logaritham scale to see difference easilyt if our
predicted values get small if our
if predictionTable['Probability'].max() == 0:
    plt.yscale('log')

plt.show()
```

```
   X (Number of Occurrences)  Probability
0                        6.3     0.005587
1                        7.3     0.011611
2                        8.3     0.021225
3                        9.3     0.034626
4                       10.3     0.051005
```



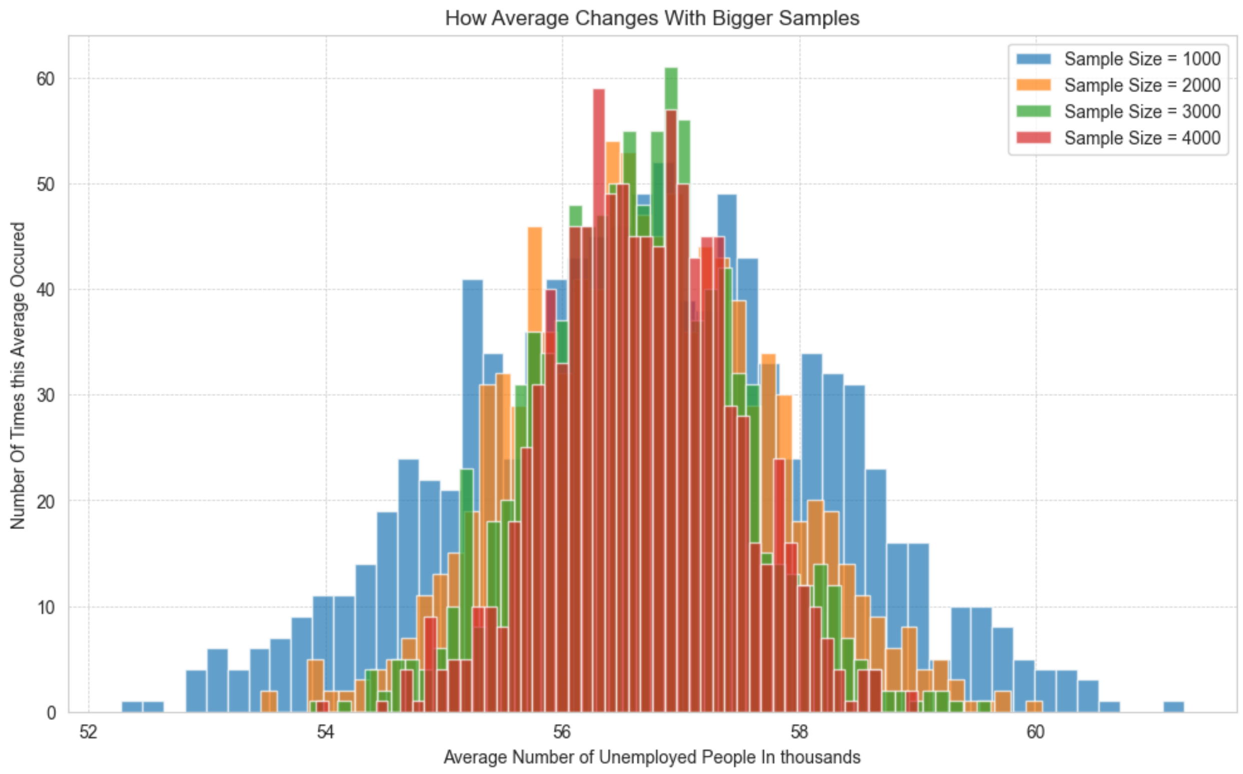
As we know that we are checking the unemployment data, so poisson model work well good for this data because it let us see teh probability or different chances of unemployemnt rate heppeniung in specific month. We made different graphs of different time to have a depeer look. because young people can not be wokring during summer time becausethey are in collges or universities teh rest of year. So with the help of poisson model here we can see a clear picture that when young people are out of job.

What Happens with Big Data?

Well I beleive that if we have alot of data points things starts becomming more predicctable. Let's check how our data behaves when we have more of it.

```
def
sampleAverage(dataOfUnemploymentInThousands,sampleSize,times=1000):
    averages=[]
    for _ in range(times):

sample=np.random.choice(dataOfUnemploymentInThousands,sampleSize)
        averages.append(sample.mean())

    return averages
```

```python
dataOfUnemploymentInThousands = monthlyUnemploymentFilteredDF['VALUE
In Thousands']
#sampleSize=[10,50,100,500]
sampleSize=[1000,2000,3000,4000]

plt.figure(figsize=(12,7))
for size in sampleSize:
    averages=sampleAverage(dataOfUnemploymentInThousands,size)
    plt.hist(averages,bins=50,alpha=0.7,label=f"Sample Size = {size}")
plt.xlabel("Average Number of Unemployed People In thousands")
plt.ylabel("Number Of Times this Average Occured")
plt.title("How Average Changes With Bigger Samples")
plt.legend()
plt.grid(True,which="both",linestyle="--",linewidth=0.5)
plt.show()
```



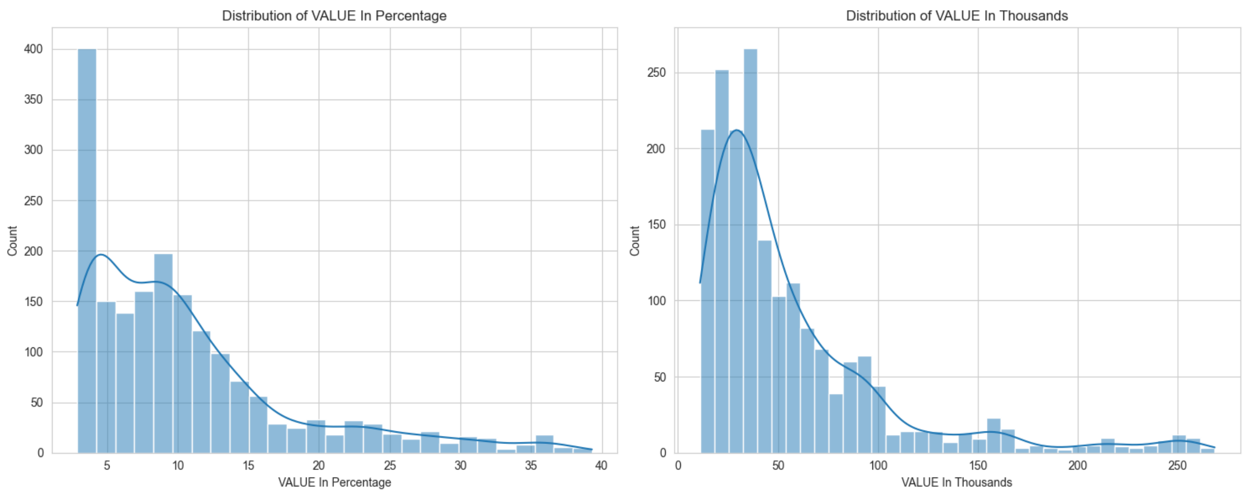
We took our unemployment number and think that what would happen if we only took piece of these numbersinstead of whole data? will the average be close to true average of all numbers. To find out we take different samples like 1000, 2000, 3000, 4000. We actually wanted to check that if our sample's average is reliable. let's understand it with a example to make it more clear. suppose we have alot of cookies and we taste one cookie from them, can we say for sure how all cookies taste? we did this thing with unemployment numbers to see how big our number need to be to get a good taste of whole data.

In the number we an see different olored bar for different size. If the bars are wider that means we are unsure about the average. And when number gets bigger the bars get closer which means we are consistant about teh aerage. Well smaller numbers's average an be bit off but bigger nubers give us clear picture. By experimenting with different numbers we got o know that bigger samples give us clear understanding of unemployment data.

Use Normal distribution to explain or identify some information about your dataset. [0-20]

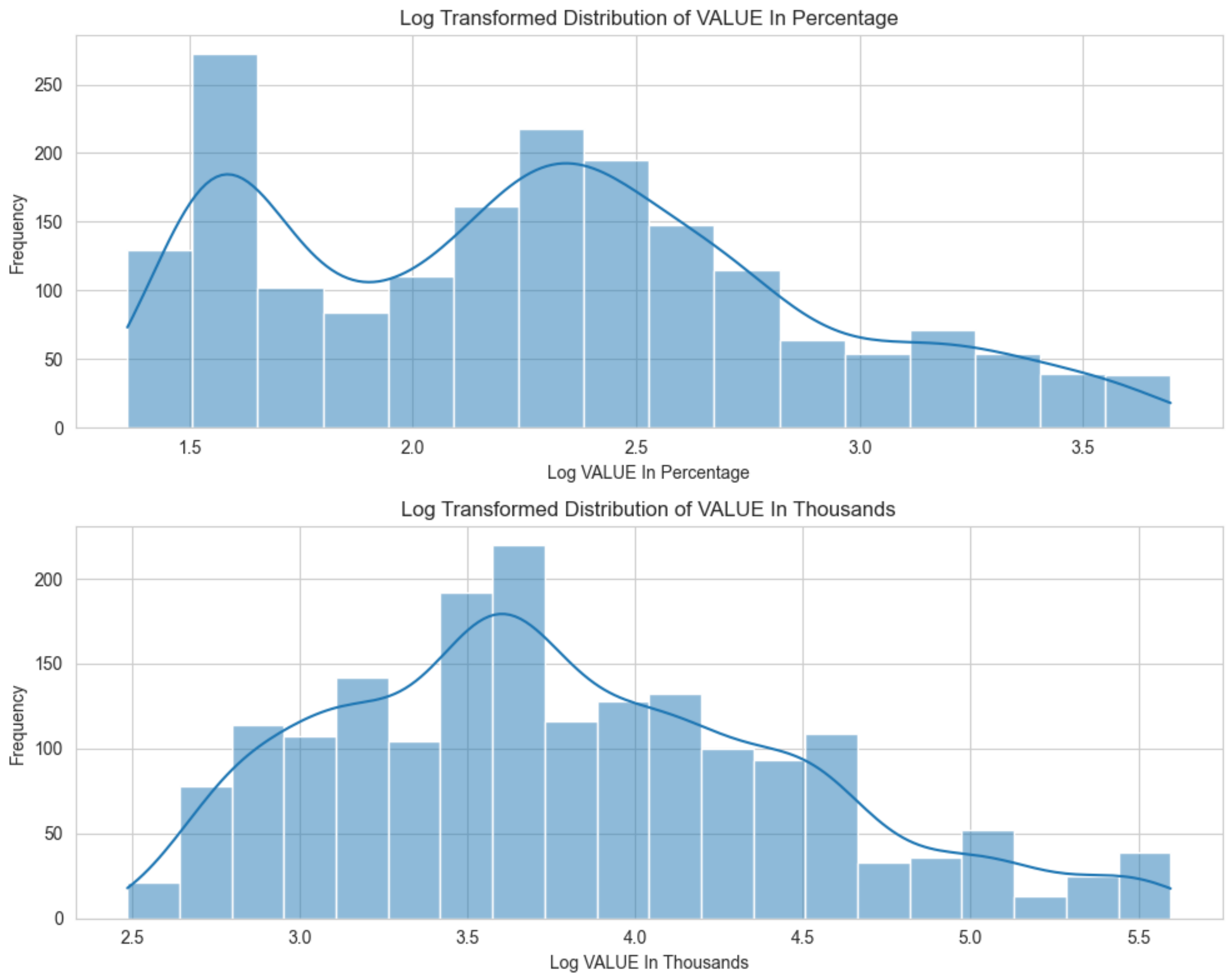
First we will check if the main column of our datase follows the normal distribution. We will use VALUE In Thousands and VALUE In Percentage to check for normal distribution. These are most relevant columns for this purpose.

```
# Let's style our plot
sbn.set_style("whitegrid")
# Now lets plot the distribution of VALUE In Thousands and VALUE In
Perentage
fig,ax=plt.subplots(1,2,figsize=(15,6))
sbn.histplot(data=monthlyUnemploymentFilteredDF,x="VALUE In
Percentage", kde=True,ax=ax[0])
ax[0].set_title("Distribution of VALUE In Percentage")
ax[0].set_xlabel("VALUE In Percentage")
ax[0].set_ylabel("Count")

sbn.histplot(data=monthlyUnemploymentFilteredDF,x="VALUE In
Thousands", kde=True,ax=ax[1])
ax[1].set_title("Distribution of VALUE In Thousands")
ax[1].set_xlabel("VALUE In Thousands")
ax[1].set_ylabel("Count")

plt.tight_layout()
plt.show()
```



They do not seem to be perfectly normal distributed. we can see that they are right skewed. This skewness suggests that or data our data values are concentrated on the ledt side with long tail on right. For right skewed distribution we can considered the logarithic nomality. This will compress large scale and spread the values. Let's apply this transformation and see the results.

```
#Lets apply the log transformation
monthlyUnemploymentFilteredDF['Log VALUE In
Percentage']=np.log1p(monthlyUnemploymentFilteredDF['VALUE In
Percentage'])
monthlyUnemploymentFilteredDF['Log VALUE In
Thousands']=np.log1p(monthlyUnemploymentFilteredDF['VALUE In
Thousands'])
# lets set up the plot layout
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(10,8))
# Now lets plot teh distribution
sbn.histplot(data=monthlyUnemploymentFilteredDF['Log VALUE In
Percentage'],kde=True,ax=ax[0])
ax[0].set_title("Log Transformed Distribution of VALUE In Percentage")
ax[0].set_xlabel("Log VALUE In Percentage")
ax[0].set_ylabel("Frequency")

sbn.histplot(data=monthlyUnemploymentFilteredDF,x="Log VALUE In
Thousands", kde=True,ax=ax[1])
ax[1].set_title("Log Transformed Distribution of VALUE In Thousands")
ax[1].set_xlabel("Log VALUE In Thousands")
ax[1].set_ylabel("Frequency")

plt.tight_layout()
plt.show()
```

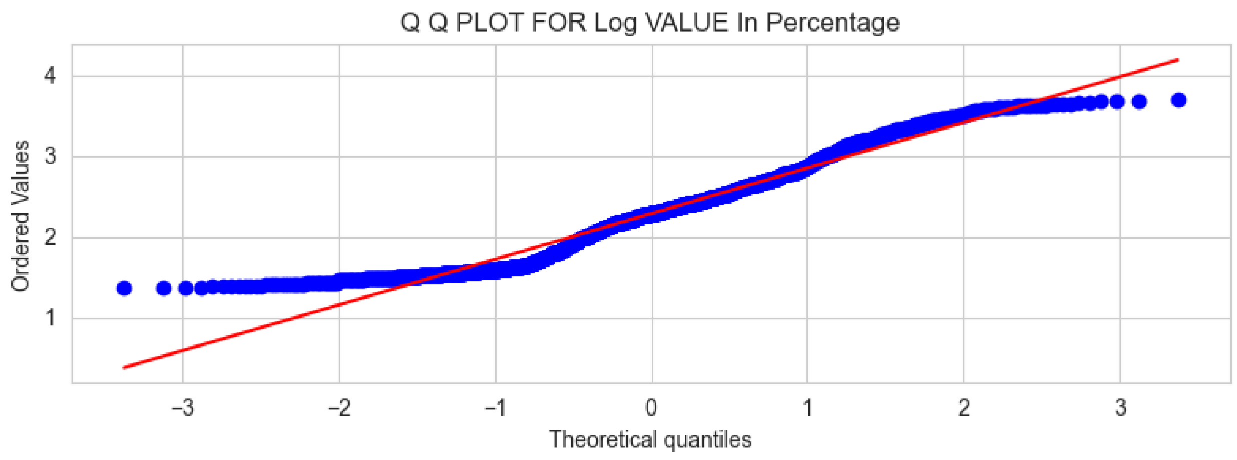


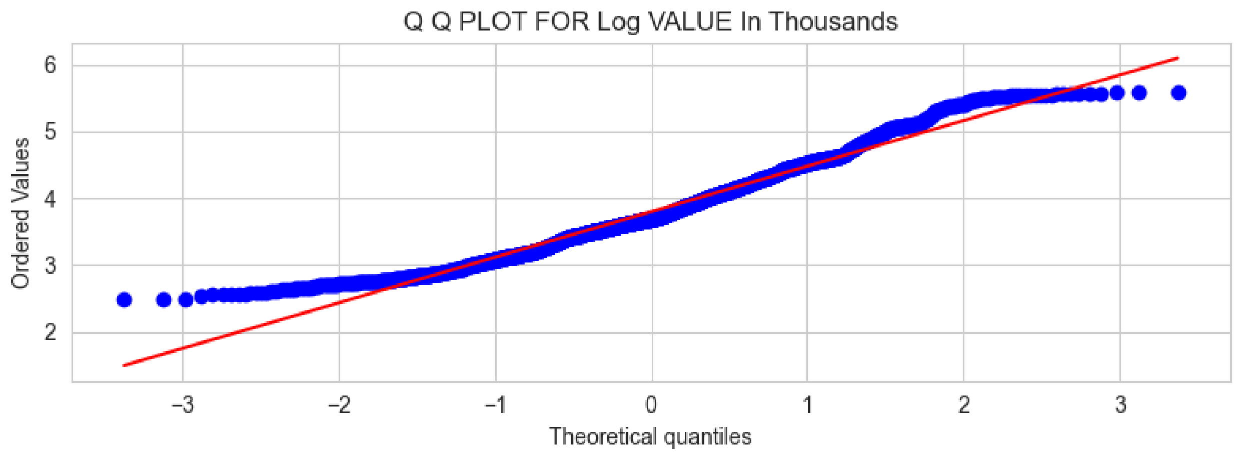
After using the logarithmic which is like marh trick to get the data into normal distribution the data looks balanced on both side. But the data is not perfectly shaped but it alost follows the bell curve.

Now let's use QQ plot to check the distribution.we will compare dataset's quantiles wirth the quantiles from standard normal distribution and if the points foloows the straigh line or cclose to that line so it means it follows the normal distribution.

```
# lets setup our plot layout
fig,ax=plt.subplots(nrows=2,ncols=1,figsize=(8,6))

# Now lets create QQ plot for VALUE In Percentage and VALUE In
Thousands
stats.probplot(monthlyUnemploymentFilteredDF['Log VALUE In
Percentage'], dist="norm",plot=ax[0])
ax[0].set_title("Q Q PLOT FOR Log VALUE In Percentage")

stats.probplot(monthlyUnemploymentFilteredDF['Log VALUE In
Thousands'], dist="norm",plot=ax[1])
ax[1].set_title("Q Q PLOT FOR Log VALUE In Thousands")
```

```
plt.tight_layout()
plt.show()
```





As we can see that the data is not stricly follows teh straigh line but it is very lose to the normal distribution.

Question: Explain the importance of the distributions used in point 3 and 4 in your analysis. Justify the choice of the variables and explain if the variables used for the discrete distributions could be used as normal distribution in this case. [0-15]

Since I like to explain things with real time examples so let's answer this question with real world example. Imagine some photographer who captures essence of dublin city. He has 2 lens in his bag, one is to zoom into some specific happenigs and other is to have broader overview. The one which is for specific events is the example of poisson and the other len is for Normal distribution.

Let's talk about poisson. This zooming lens help us to focus on a specific event and understand it's frequency. For example if we notice a pattern like the TFI Bus No 15 which arrives after every 10 minutes so the poisson distribution helps capture this consistency.

In our dataset the columns VALUE In Thousands and VALUE In Percentage represents the specific event like the number of time value appears. so with the help of poisson distribution we can understand how regularly these alues appear in specific time period. It is valuable because it helps us see patterns and ake predictions and understands more about our data that what it is trying to tell.

As I said above that Normal distribution captures broader overview. As we were talking about a speific bus in poisson but in Noranl we will say that we will capture the whole trafficsystem of city during different times of day. So we can get general overview. With poisson we looked at specific data details and then saw bigger picture of our data using normal distribution. With these both views we get the complete picture of our data.

Also we can use the same ariables for both distributions. The values in "VALUE In Percentage" and "VALUE In Thousands" shows frequencies. poisson distribution captures exact counts for specific periods. And the noral distribution an help understand the general trend of these froma broader view. And sometimes data dosen't fit the normal distribution by default so we used some tehniques to get it into normal distribution. We used the log to transform our data to normal distribution.

Machine learning for Data Analytics

Question: Explain which project management framework (CRISP-DM, KDD or SEMMA) is required for a data science project. Discuss and justify with real-life scenarios. Provide an explanation of why you chose a supervised, unsupervised, or semi-supervised machine learning technique for the dataset you used for ML modeling.[0-20]

If we look into the process we will be doing for Machine learning we will be training and comparing different supervised techniques like regerssion and classification. So I think CRISP-DM would be good framework. It guides through understanding the problem and prepares the data and models and then evaluate results. We can say it is like a cyclic process that allows refinement at every stage. Lets understand this with some example, suppose we have a challange of baking cake and we get a kitchen to prepare it and in kitchen we get some recepies and and then we have judges who will be tasteting it. In this example the kitchen where we have our recepies is our data and the recepies are our machine learning models that we will be trying and the judge are our results that we will be checking.

And now just instead of putting everything to oven we need a plan that will help us to step through the peocess, and make adjustments and we will end up making the best cake. So this is where CRISP-DM comes. it works like a recepie book for success in data project.

So lets dig in it more and learn what it can help uS. First of all it helps us to understand the challenge just like understanding we need choclate or vanilla cake in our baking cake problem. CRISP-DM help us define the problem we solving with our data.

And after deciding which flavour cake we need, we know about our ingrediants. CRISP helps us exporing the data to check that what data do we have. So we will explore the data and know about it that what we are working with.

Then after deciding our ingrediants we mix flour and melt butter and in CRISP case we make sure that we do all teh necessary work just like cleaning of our data and getting it ready for work.

After that we try recepies. We use different ingrediants to see which gets best results. Sae in data, we use different models to check which one is best.

Then evaluatiion comes at teh end, when cake gets out of oven we taste it. In data term we can say we evaluate our models and then we check which one is the winner from all.

So our project is like baking cake challenge. we tried different models to find the best one. This is god because trusting just one model we need to try different andfind best one.

Let's take a real lifde example, suppose a bank wants to predict likelihood of loan default. CRISP-DM framework would guide the project from undertsanding the business objective (which will be reuducing the default rate) to deploying a predictive model that will access risk of default for new applicants.

Let's take another example, suppose a retail company can use CRISP-DM to predict inventry demand. so it will be start with understanding the business goal and then data collection and prepration, then building a model and then evaluation of predictions.

For our Machine learnign tasks we used supervised learning as we have labeled data. so in our datasets we had the data where outputs are known so in such cases teh supervised learnign is teh best example.

And suppose if we have a project where data is without predefined labels , for example identify customer segments or we can say finding some anomalies which are unusual pattern, tehn unsupervised learning can be used. it is just like exploring without a map. In semi supervised we get some labeled and some unlabeled data.

Question: Machine learning models have a wide range of uses, including prediction, classification, and clustering. It is advised that you assess several approaches (at least two), choose appropriate hyperparameters for the optimal outcomes of Machine Learning models using an approach of hyperparameter tunning, such as GridSearchCV or RandomizedSearchCV. [0 - 30]

Show the results of two or more ML modeling comparisons in a table or graph format. Review and critically examine the machine learning models' performance based on the selected metric for supervised, unsupervised, and semi-supervised approaches. [0 - 30]

So we will be using different models and classifiers below and after that we will write their explanation and will talk anout supervised , unsupervised, and semi supervised approaces and compare the results using tables as asked in questions. so wil be answering both of the above questions.

When workinh with data different columns can have different ranges. if ranges are big so machine learning models won't work good. results could be biased. To solve this issue we adjust the data so every column has values in same range. We are using StandardScaling which is used to scale our data. it adjust values in every olumn so average is zero. So we will be using some columns from our dataset as features and and one specific column as targert which we will predict.

```
# Lets split data into features and target as i discussed above
#X = monthlyUnemploymentFilteredDF.drop(columns=['Log VALUE In
Percentage','Log VALUE In Thousands', 'Statistic Label'])

selectedFeatures = [ "Age Group_15 - 24 years", "Age Group_25 - 74
years","Year", "MonthNumber","Sex_Male", "Sex_Female"]

# Splitting the data into training and testing sets (80% train, 20%
test)
X = monthlyUnemploymentFilteredDF[selectedFeatures]
#X = monthlyUnemploymentFilteredDF.drop(columns=["Age Group_15 - 24
years", "Year", "Sex_Male", "Sex_Female"])
y=monthlyUnemploymentFilteredDF['Log VALUE In Percentage']
# Now lets split the data into training and testing
X_train,
X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=
42)
# initialize the object using StandardScaler
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)

# it will display the first 5 rows of training data
X_train_scaled[:5]
```

```
array([[-0.99394952,  0.99394952, -0.18404376, -0.40817474,
1.39162986,
        -0.69925376],
       [-0.99394952,  0.99394952, -1.26498196, -0.40817474,
1.39162986,
        -0.69925376],
       [-0.99394952,  0.99394952,  0.49154262,  1.6172851 , -
0.71858188,
         1.43009599],
       [-0.99394952,  0.99394952,  1.30224627,  0.74923088,
1.39162986,
        -0.69925376],
       [-0.99394952,  0.99394952,  0.22130807,  1.6172851 , -
0.71858188,
         1.43009599]])

# Lets create object of LinearRegression
linearRegressionModel=LinearRegression()
linearRegressionModel.fit(X_train_scaled,y_train)
#lets predict on test set
linearRegressionPredictions=linearRegressionModel.predict(X_test_scale
d)
# Now we can calculate the Mean Squared Error for our Linear
Regression Model
linearRegressionMSE=mean_squared_error(y_test,linearRegressionPredicti
ons)
linearMAEValue = mean_absolute_error(y_test,
linearRegressionPredictions)
linearR2 = linearRegressionModel.score(X_test_scaled, y_test)
print(linearRegressionMSE)

# lets create a dataframe to show the actual and predicted values
comparisonDF=pd.DataFrame({"Actual Outcomes: ": y_test, "Predicted
Outcomes: ":linearRegressionPredictions})
print(comparisonDF.head(10))

0.01678037620251708
      Actual Outcomes:   Predicted Outcomes: 
233            1.526056              1.657797
450            2.261763              2.569630
1240           2.341806              2.040048
1693           2.653242              2.935810
411            1.589235              1.778701
1336           2.163323              2.065668
1526           2.424803              2.751019
1222           2.379546              2.041921
1233           2.292535              1.969817
1436           2.572612              2.735733
```

The Mean Squared Error is 0.0167 whih is good. Lower MSE tells that model is good fit to data.As we can see the actual outcomes and predited outcoms as well.

```
# lets initialize teh SVR objet so we an use methods from it
svr = SVR()

# Parameters for hyperparameter tuning
parameterGridSVR = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Setup GridSearchCV for SVR
gridSearchSVR = GridSearchCV(estimator=svr,
param_grid=parameterGridSVR ,
                               cv=2, n_jobs=-1,
scoring='neg_mean_absolute_error', verbose=1)

# Perform grid search
gridSearchSVR.fit(X_train_scaled, y_train)  # Using scaled data for
SVR

# Best parameters from the grid search
bestParameterSVR = gridSearchSVR.best_params_
estimatorSVR = gridSearchSVR.best_estimator_

# Evaluate the best estimator on the test set
svrpredictions = estimatorSVR.predict(X_test_scaled)
svrMAEValue = mean_absolute_error(y_test, svrpredictions)
svrR2 = estimatorSVR.score(X_test_scaled, y_test)

svrMSE=mean_squared_error(y_test,svrpredictions)

print(bestParameterSVR, svrMAEValue ,svrR2,"\n \n")

#y_test  , svrpredictions
comparisonSVRDF=pd.DataFrame({"Actual Outcomes: ": y_test, "Predicted
Outcomes: ":svrpredictions})
print(comparisonSVRDF.head(10))
```

```
Fitting 2 folds for each of 12 candidates, totalling 24 fits
{'C': 10, 'gamma': 'scale', 'kernel': 'rbf'} 0.14891425593309485
0.8828187859726744


      Actual Outcomes:   Predicted Outcomes: 
233           1.526056              1.537231
450           2.261763              2.236655
1240          2.341806              2.361593
1693          2.653242              2.553717
411           1.589235              1.451448
1336          2.163323              2.238991
1526          2.424803              2.580013
1222          2.379546              2.347168
1233          2.292535              2.343367
1436          2.572612              2.765736
```

As we can see that the R2 is 0.8828 which is very good. And the Mean Absolute Error is 0.1489. We used SVR here and we know that it is very good at finding pattern and it can easily find complicated patterns as well. But we wanted to that our SVR gives it's best so we used a process which wecan say is like a taste test for SVR settings. This process is called hyperparameter tuning. SVR tried a lot of options for example level of flexibilities (C value) and different ways to look at it (Kernel). after that we cheked that which setting made the closest prediction. So it tried different combinations and found the best. So with these settings then SVR predicted the target good and the Mean Absolute Error is small. so if Mean Absolute Error gets small it means the predictions SVR did are clsoe to the truth. And R2 value is 0.8828 that means predictions matched from real data. So with teh help of picking right settings this model did good job on data.

Lets compare the values in a table.

```
comparisonForRegression={
    "Metric": ["Mean Squared Error","Mean Absolute Error","R2"],
    "Linear Regression": [linearRegressionMSE,linearMAEValue,
linearR2],
    "SVR": [svrMSE, svrMAEValue ,svrR2]

}
comparisonDataForRegression=pd.DataFrame(comparisonForRegression)
comparisonDataForRegression
```

```
                Metric  Linear Regression       SVR
0   Mean Squared Error           0.167804  0.035177
1  Mean Absolute Error           0.347698  0.148914
2                   R2           0.441013  0.882819
```

As you can see above that we applied two regressions. Linear Regression is good strart because it is very simple and interpretable. Then we explored SVC which we know is use to capture complex relationships than linear regression. We used GridSearchCV with Support Vector because it tune the model to find best possible parameters which leads to best performance

The results of these models like R2, MSE and MEA help us to understand that how good our model woked with the data and how good are prediction outouts are. we know that if we get lower MSE and MAE so it is better. R2 tells about variance in dependant variable which is also a way to tell how good model is at predicting.

In supervised learning we learn by looking at examples which we already know the correct answers of (like labels). The models we used linear regression and Support vector regression are part of supervised learning because we trained our models showing the rights to them. for example we had data with what the outcomes was like what is the unemployment rate of people of different age group. we used our data to get predictions from our model. I also created a table after applying both of models so we can compare the values of each model.

In unsupervised learning it is like to learn without the teacher. We just get questions without answers. so in such situations model check patterns and make sense of data on its own. But we did not use any unsupervised learning here.

Semi suppervised is the mixture of both. In semi supervised we don't know all the answers and we use the answers we have and we learn from the answers we have and guess the rest of the answers.

So in short we used the supervised learning because the models we used they learnt from data where output was already known.

Lets apply classification on this dataset as well. Before that lets get our data ready for classification. Lets create a new column named UnemploymentCategory. Right now we just have numbers in our dataset and by creating this column we are categorizing these numbers into groups. We are doing this because classifications needs to learn how to sort these number.

```
# Lets craete a new DataFrame for classification. We can create a copy of our adta and ceate a new column for classification
monthlyUnemployedDFForClassification = monthlyUnemploymentFilteredDF.copy()

def categorizeUnemployment(rate):
    if rate > 30:
        return "High"
    elif rate >=21:
        return "Medium"
    elif rate >= 11:
        return "Low"
    else:
        return "Very Low"
# Lets use apply function to column name 'Log Value In Percenatge' to
create a new column of unemploymentCategory
monthlyUnemployedDFForClassification['UnemploymentCategory']=monthlyUn
employedDFForClassification['VALUE In
Percentage'].apply(categorizeUnemployment)

# Lets check how our dataset looks now
monthlyUnemployedDFForClassification.head()
```

```
                                 Statistic Label  VALUE In Percentage
\
0  Seasonally Adjusted Monthly Unemployment Rate                 13.3

1  Seasonally Adjusted Monthly Unemployment Rate                 13.7

2  Seasonally Adjusted Monthly Unemployment Rate                 12.7

3  Seasonally Adjusted Monthly Unemployment Rate                  7.4

4  Seasonally Adjusted Monthly Unemployment Rate                  6.9


   VALUE In Thousands  Year  MonthNumber  Age Group_15 - 24 years  \
0                55.4  1998            1                        1
1                31.5  1998            1                        1
2                23.9  1998            1                        1
3                96.2  1998            1                        0
4                55.4  1998            1                        0

   Age Group_25 - 74 years  Sex_Both sexes  Sex_Female  Sex_Male  \
0                        0               1           0         0
1                        0               0           0         1
2                        0               0           1         0
3                        1               1           0         0
4                        1               0           0         1

   Log VALUE In Percentage  Log VALUE In Thousands
UnemploymentCategory
0                 2.660260                4.032469
Low
1                 2.687847                3.481240
Low
```

```
2                  2.617396                3.214868
Low
3                  2.128232                4.576771             Very
Low
4                  2.066863                4.032469             Very
Low
```

```
# Now lets prepare the feature and target variable
X = monthlyUnemployedDFForClassification[['VALUE In Percentage']]
y=monthlyUnemployedDFForClassification['UnemploymentCategory']

# now let's split the data into training and testing
X_train,X_test,y_train,y_test,=train_test_split(X,y,test_size=0.3,rand
om_state=42)

#initialize random forest object
randomForestClassification=
RandomForestClassifier(n_estimators=100,random_state=42)

# now lerts train our model
randomForestClassification.fit(X_train,y_train)
# Now lets predict on test
yPrediction=randomForestClassification.predict(X_test)
# Now lets check teh accuracy
accuracySimpleRFClassifier=accuracy_score(y_test,yPrediction)
classReportRFClassifier=classification_report(y_test,yPrediction)
print(accuracySimpleRFClassifier)
print(classReportRFClassifier)
```

```
1.0
              precision    recall  f1-score   support

        High       1.00      1.00      1.00        19
         Low       1.00      1.00      1.00       133
      Medium       1.00      1.00      1.00        36
    Very Low       1.00      1.00      1.00       369

    accuracy                           1.00       557
   macro avg       1.00      1.00      1.00       557
weighted avg       1.00      1.00      1.00       557
```

Now lets apply support vector classfdier and check its rewsult then we can comapre both of the results in a table.

```
# Now lets prepare the feature and target variable
X = monthlyUnemployedDFForClassification[['VALUE In Percentage']]
y=monthlyUnemployedDFForClassification['UnemploymentCategory']
```

```
# now let's split the data into training and testing
X_train,X_test,y_train,y_test,=train_test_split(X,y,test_size=0.3,rand
om_state=42)

#initialize random forest object
supportVectorClassification= SVC(gamma="auto")

# now lerts train our model
supportVectorClassification.fit(X_train,y_train)
# Now lets predict on test
yPredictionSVC=supportVectorClassification.predict(X_test)
# Now lets check teh accuracy
accuracySimpleClassifierSVC=accuracy_score(y_test,yPredictionSVC)
classReportClassifierSVC=classification_report(y_test,yPredictionSVC)
print(accuracySimpleClassifierSVC)
print(classReportClassifierSVC)
```

```
0.9946140035906643
              precision    recall  f1-score   support

        High       1.00      1.00      1.00        19
         Low       0.98      1.00      0.99       133
      Medium       1.00      1.00      1.00        36
    Very Low       1.00      0.99      1.00       369

    accuracy                           0.99       557
   macro avg       0.99      1.00      1.00       557
weighted avg       0.99      0.99      0.99       557
```

```
# Lets Generate classification reports
classReportRFClassifier = classification_report(y_test, yPrediction,
output_dict=True)
classReportClassifierSVC = classification_report(y_test,
yPredictionSVC, output_dict=True)

# Now lets create DataFrame for random forest classifier report
randomForestReportForComparison =
pd.DataFrame(classReportRFClassifier).transpose()
svcReportForComparison =
pd.DataFrame(classReportClassifierSVC).transpose()

# Lets now create Data frame for SVC
randomForestReportForComparison["Classifier"] = "Random Forest"
svcReportForComparison["Classifier"] = "Support Vector Classifier"

# lets add together both of dataframes
comparisonForClassifierDF =
pd.concat([randomForestReportForComparison, svcReportForComparison],
axis=0).reset_index()
```

```
comparisonForClassifierDF.rename(columns={'index': 'Class'},
inplace=True)
finalComparisonForClassification =
comparisonForClassifierDF[["Classifier", "Class", "precision",
"recall", "f1-score", "support"]]
finalComparisonForClassification
```

```
                   Classifier         Class  precision    recall  f1-
score  \
0               Random Forest          High   1.000000  1.000000
1.000000
1               Random Forest           Low   1.000000  1.000000
1.000000
2               Random Forest        Medium   1.000000  1.000000
1.000000
3               Random Forest      Very Low   1.000000  1.000000
1.000000
4               Random Forest      accuracy   1.000000  1.000000
1.000000
5               Random Forest     macro avg   1.000000  1.000000
1.000000
6               Random Forest  weighted avg   1.000000  1.000000
1.000000
7   Support Vector Classifier          High   1.000000  1.000000
1.000000
8   Support Vector Classifier           Low   0.977941  1.000000
0.988848
9   Support Vector Classifier        Medium   1.000000  1.000000
1.000000
10  Support Vector Classifier      Very Low   1.000000  0.991870
0.995918
11  Support Vector Classifier      accuracy   0.994614  0.994614
0.994614
12  Support Vector Classifier     macro avg   0.994485  0.997967
0.996191
13  Support Vector Classifier  weighted avg   0.994733  0.994614
0.994633

       support
0    19.000000
1   133.000000
2    36.000000
3   369.000000
4     1.000000
5   557.000000
6   557.000000
7    19.000000
8   133.000000
9    36.000000
10  369.000000
```

```
11      0.994614
12    557.000000
13    557.000000
```

Now lets compare these 2 classifications

When we talk about different machine learning models we see how each one performs and predicts. It is like different people of same profession are doing a thing and eeryone will do as per his knowledge and experience so the results will be different of for each one. In our notebook we used Support Vector Classifier and Ranndom Forest classifier on the same data to classify and predict . After getting results we checked both teh predictions and compared the accuracy.

These classifier helped us to make sense of information. They categorized very good. Random forest classifier gave accuracy of 1.0 which is accuracy of 100 percent and the support verctor classifier gave accuracy of 0.99 which is also very good.

SVC and Random Forest both are supervised learning models. As we know supervised learning models are trained using label data. They learn to map input data to known output lables. And SVC and Random Forest needs input output pair to learn. so these are supervised learning.

Demonstrate the similarities and differences between your Machine Learning modelling results using the tables or visualizations. Provide a report along with an explanation and interpretation of the relevance and effectiveness of your findings. [0 - 20]

Lets analyze the regression that we did on unemployment Dataset. We were predicting the unemployment rate on some variables. The matrix we considered are:

1) MSE: Mean Squared Error tells us how much predictions vary from actual value on average. If we get low MSE that means it is good which means that our predictions are close to the data.

2) MAE: Mean Absolute Error is the average absolute difference between prediction and real value.

3) R2: R2 score tells that how data fit the regression model.

From regression analysis we learnt that the models predicted the unemployment rate with good accuracy. Support Vector Regressor adjusted better to data than linear regression model and we can see that in comparison table.

For classificaion we used the Support Vector Classifier and Random Forest. As wrote earlier that these models are just like 2 different person of same profession with their experience and knowledge. As we can see that both did a good job and the resukts we got are very perfcet. These results we got fromour models are important. because they tell us that how uch we can trust them with unemployment data.

I already displayed the comparison table above after each model.

Question: Show the results of two or more ML modeling comparisons in a table or graph format. Review and critically examine the machine learning models' performance based on the selected metric for supervised, unsupervised, and semi-supervised approaches

First of all let's understand the Machine Learning Models. The linear Regression tries to draw a line which fits the data points. for example we have points on graph and we want to draw line through them. this is what this model does.

And the random forest is like a group which makes decision together. we can say like a group of trees. so every tree has their own opinion and they give their opinion and at the end the majority decision gets taken. This way we can get more accurate decision.

If we compare machine learning models so we can say linear model is easy which is like using simple calculator and random forest is like using scientific one. linear model is ofcouse easy to use but random forest can do complex things. When it comes to result so random forest gives more accurate results because it takes different opinions. So we can say that Random forest is versatile but sometimes it over complicates.

We also used Random Forest and Support Vector Classifier for our data. both performed very good. i already diaplyed the comparison graph above. Random Forest classifier has precision,recall,f1-score,support of 1.0 whch is 100 percent and Support vector has 0.99.

I have already displayed the cluster graph above.

Programming for DA

Question:1. The project must be explored programmatically, this means that you must implement suitable Python tools (code and/or libraries) to complete the analysis required. All of this is to be implemented in a Jupyter Notebook. Your codebook should be properly annotated. The project documentation must include sound justifications and explanation of your code choices (code quality standards should also be applied). [0-50]

This jupyter notebook is used to perform all the assigments using pyhton. This notebook is a comprehensive and systemic approach to understand and predict unemployment rates over time.

We started with loading all the required python libraries for example pandas, numpy which used for data manipulation, cleaning of data, and statistical analysis. Then matplotlib and seaborn used for data visualization, and to understand unemployment trends and distribution.

scipy.stats used for statistical analysis. scikit-learn is used for machine learning and for providing tools for data scaling and splitting and modeling.

Then we loaded the dataset using the pandas standard structure. Then we used the initial EDA methods like head(), tail(), shape(), and column to get the understanding of our data struture. Then we checked for data cleaning like null values, duplicate values. by this way we ensure the dats's quality and readability. Then info() method is used to check for data types. and we found that Month column is in string type so we converted it into datetime. After taht we used the describe method to fund the overview of central tendencies and distributions of our dataset.

After that I have applied different visualizations like histogram, line plots, box plots, were generated to understand the distribution,trends and relationships in our data. Each plot is labeled and titled properly.

After that i have applied feature engineering such as year and month from date column for time based analysis. Then categorical features were encoded to convert for suitable format so it will be good for machine learning models.

Then i have compared the distribution with theorethical distribution (Poisson). And applied normal distribution to our data.

Then I have applied different regression models in ML after preparing data for ML. so we predicted unemployment rates. Each model choice is chossed by good reasoning. I also applied clustring on my data. so first i scaled the data and then clustered it using Kmean. And tabled the model performance.

The entire notenook is properly annotated with markdown cells and explaining the purpose. This way notebook gets easily understandable and even a non technical person can understands it. In conclusion the project is explored programaticcaly using a varitey of python tools. ensuring a detailed systematic and comprehensive analysis. Every step from laoding the data to performing ML modeling was carefully chosen and justified. Code quality standards and providing a holistic understanding of unemployment trends

One more thing I have used camel notation to write variables in my notebook. As we know variables can't be keywords and if it is long(combination of different words so we can use '_' in them). it is not required to put underscore in them. this way variables become just more readable. but instead of putting underscore in multiple words i used camel notation because most of classes use this practice that they put underscore to seprate different words. for example if we talk about Support Vector Model. we see functions like that 'mean_squared_error' . and most of the classes like support vector use this practice. so just to not get in confussion i choosed the camel notation and used it throuout in my notebook.

I have also expalined each code by using Mrkdown after outout that what we did and why we did and what we gained from this result.

Question: Briefly discuss your use of aspects of various programming paradigms in the development of your project. For example, this may include (but is not limited to) how they influenced your design decisions or how they helped you solve problems. Note that marks may not be awarded if the discussion does not involve your specific project [50]

Let's discuss different programming paradigms i have used in this project. paradigm is nothing just a way viewing the world. Look at this notebook, You will see things in a sequence like loading of data and then cleaning of data and then analyzing, visualizing and then modeling. so we can say it is a procedural style. steps are performed in an order. It is like if we cook pasta so we follow a recepie. Same in this project I have used the order because each step is dependent on previous one. By using this our notebooks gets in logical flow. it get's like a story book. Also it gets simple to understand and wasy to follow and even if someone revsit it an understands it easily.

I have used different methods in this notebook. Like I have created different DataFrames and used Head(), isnull() and other methods. and to use these methods directly on the object. use of these models like sikit-learn leverages to Object Oriented Programming. Like in object oriented programming langauges we craete the objects first and then use the methods of that object using dot(.) operator. In this Notebook we are doing that as well. For instance to use standardScaler in our notebook i first initialiezd the StandardScaler like this "standardScalerobj= StandardScaler()" and then used standardScalerobj to call the methods of StandardScaler class.

The benefit of using OOP in the project is that it alows us to reuse the code and the structure gets clear especially when we dealing with visualizing and odeling of data. And it also gets easy to manipulate the data. One thing i love about OOP is every related thing gets packed together. like StandardScaler functions will be just in that class.

Then functional programming is also involved. simple example of this is use of functions from different libraries, for eample train_test_split and scipy which we used for different statistical functions. well these functions takes arguesnta and return processed arguents. for example I have used this in the project "train_test_split(X, y, test_size=0.2, random_state=42)". this function is taking arguents to split the data into testing and training. arguments are in brakects "()". then this function will process the data given in the bracekts and returned processed data. these are ready-made tools to use to do some tasks.

Then in this notebook we have filtered our data multiple times. for example we used this "monthlyUnemploymentDF[monthlyUnemploymentDF['Sex']=='Both sexes']" here we are filtering our data using equal opeartoe. so this goes into declarative programming. declarative programming is like when we write code that describe what we want to do rathan than how we wanna do. like in the above code we are filtering sex column. With the help of this things gets clear. like i said in the code line that only consider those rows where genser is "Both Sexes". With the help of these techniques notebook gets easy to understand for anyone.